Demetrius Costa Silva Faria Lima

Soluções Alternativas de Escalonamento e Gerenciamento de Caminhos para o MPTCP

Itajubá – MG

Outubro de 2017

Demetrius Costa Silva Faria Lima

Soluções Alternativas de Escalonamento e Gerenciamento de Caminhos para o MPTCP

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação.

Universidade Federal de Itajubá – UNIFEI

Programa de Pós-Graduação em Ciência e Tecnologia da Computação – POSCOMP

Orientador: Prof. Dr. Bruno Yuji Lino Kimura

Itajubá – MG

Outubro de 2017

AGRADECIMENTOS

Agradeço a Deus, a minha esposa Júlia, familiares e amigos. Principalmente, agradeço ao meu orientador prof Bruno Kimura pelo direcionamento, conselhos e apoio neste trabalho.

Aos Profs. Bruno Khuene e Enzo Seraphim, que viabilizaram a implantação do ambiente experimental com o empréstimo das *workstations* HP.

Ao Prof. Edmilson Marmo, pelo acompanhamento, enquanto coordenador do POSCOMP.

Ao Prof. Guilherme Bastos, pelo acompanhamento, como atual coordenador do POSCOMP.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte financeiro através da bolsa de mestrado.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) que, através do projeto #2015/18808-0 coordenado pelo orientador Prof. Bruno Kimura no ICT/UNIFESP, viabilizou a realização dos experimentos com placas de rede HP *quad-port*.

“Any sufficiently advanced technology is equivalent to magic.”
(Sir Arthur C. Clarke)

RESUMO

O MultiPath TCP (MPTCP) é um conjunto de extensões para o protocolo TCP que viabiliza a transmissão confiável sobre múltiplos caminhos fim-a-fim entre cliente e servidor na Internet. Dois benefícios são imediatamente obtidos com o protocolo: $i)$ o aumento da vazão fim-a-fim através da agregação da transmissão por múltiplos sub-fluxos TCP; $ii)$ maior tolerância a falhas de comunicação, uma vez que, na indisponibilidade de um sub-fluxo, há um ou mais sub-fluxos ativos. Neste trabalho, investigamos experimentalmente dois mecanismos internos do MPTCP: **escalonamento de pacotes**, que determina a escolha do sub-fluxo no momento da transmissão; **gerenciamento de caminhos**, que determina a forma como os sub-fluxos são estabelecidos. Identificamos problemas de desempenho envolvidos em cada um desses mecanismos e propomos soluções alternativas. O primeiro problema é resultante do escalonamento padrão do MPTCP, *Lowest Latency* (LL), cujo critério de escolha de sub-fluxo de envio é definido exclusivamente pela menor estimativa dos tempos observados de ida e volta dos pacotes, podendo levar a decisões inconsistentes quando sub-fluxos estão sob perda de pacotes em caminhos congestionados. Em contrapartida ao LL, foi proposto e implementado três escalonadores alternativos – *Largest Window Space* (LWS), *Lowest Time/Space* (LTS) e *Highest Sending Rate* (HSR), que utilizam parâmetros mais confiáveis para construir critérios de decisão mais consistentes. O segundo problema está envolvido com o gerenciador de caminhos padrão do MPTCP, denominado *Full-mesh*, que estabelece um número excessivo de sub-fluxos, gerando concorrência entre próprios sub-fluxos que compartilham um mesmo caminho fim-a-fim e, consequentemente, limitando a vazão fim-a-fim. Para solucionar esse problema, foi proposto e implementado um novo gerenciador de caminhos – *Single-mesh*, que é capaz de mitigar o congestionamento inter-caminho, aumentando a taxa de transmissão dos sub-fluxos em caminhos não congestionados. Um ambiente experimental foi implantado em laboratório e, então, projetado e realizado experimentos exaustivos para avaliar as soluções propostas. Em grande parte dos experimentos, foram obtidos resultados satisfatórios, de melhor desempenho, ao comparar as soluções propostas com as soluções padrão existentes no MPTCP. Em caminhos de capacidades heterogêneas e maiores taxas de perda de pacotes, as soluções alternativas de escalonamento proporcionaram ganhos de até 7 % na vazão fim-a-fim em comparação ao LL, inclusive sobre diferentes mecanismos de controle de congestionamento do MPTCP. O gerenciamento de *Single-mesh* mostrou ser eficiente para alguns controles de congestionamento, obtendo mais que o dobro na vazão de *Full-mesh* sobre caminhos de capacidade heterogênea não congestionados.

Palavras-chave: MultiPath TCP (MPTCP). Escalonamento de Pacotes. Gerenciamento de Caminhos.

ABSTRACT

Multipath TCP (MPTCP) is a set of extensions for TCP to allow realible transmissions over multiple end-to-end paths between client and server on the Internet. Two benefits are forthwith achieved with MPTCP: i) the increase of end-to-end throughput through the transmission aggregation from multiple TCP subflows; ii) greater tolerance of comunication failures, once under the unavailability of a subflow, there exists one or more active subflows to continue transmissions. In this work, we investigated experimentally two internal MPTCP mechanisms: **packet schedulling**, which selects a best subflow in the moment of transmission; **path manager**, which defines the manner the subflows are established. We identified performance problems in both mechanisms and we proposed alternative solutions. The first problem is result of the default MPTCP scheduler – Lowest Latency (LL), whose criterion of choice of a sender subflow is based exclusively on the lower estimatives of round-trip time of observed packets. Such a strategy might lead to inconsistent decisions when the subflows are experiencing packet loss over congested paths. In this context, we propose and implement three alternative schedulers – Largest Window Space (LWS), Lowest Time/Space (LTS), and Highest Sending Rate (HSR), which make use of reliable parameters to enable criterions of choice more consistant than LL. The second problem is involved with the default MPTCP path manager – Full-mesh, which establishes an excessive amount of subflows. Such a strategy increases concurrency between the own subflows that share the same end-to-end path and, hence, it strangulates the multipath throughput. To solve this problem, we propose and implement a new path manager – Single-mesh, which is capable of mitigating inter-path congestion, while increasing the throughput of subflows over not-congested paths. We deployed a lab network setup, then we design and execute exhaustive experiments to evalute the proposed solutions. In most experiments, we obtained satisfactory results, with greater peformance when comparing the propored solutions with the default ones deployed in MPTCP. On paths of heterogeneous capacity and higher packet loss rates, the proposed alternative schedulers provided gains of up to 7 % of throughput compared to LL, even on different MPTCP congestion control mechanisms. Single-mesh manager proved to be efficient in some of the existing congestion controls, obtaining more than double of Full-mesh's throughput over not-congested paths of heterogeneous capacity.

Keywords: Multipath TCP (MPTCP). Packet Scheduling. Path Management.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AIMD	Additive Increase Multiplicative Decrease
BALIA	Balanced Linked Adaptation Algorithm
CP	Constraint-based Proactive
DSN	Data Sequence Number
FDPS	Forward-Delay-based Packet
FM	Full-mesh
HMAC	Hash-based Message Authentication Code
HSR	Highest Sending Rate
LIA	Linked-Increases Algorithm
LL	Lowest Latency
LTS	Lowest Time/Space
LWS	Largest Window Space
MPTCP	MultiPath TCP
MSS	Maximum Segment Size
MTU	Maximum Transmission Unit
NAT	Network Address Translation
OLIA	Opportunistic Linked-Increases Algorithm
QoS	Quality of Service
RDR	Redundant Data Re-injection
RR	Round Robin
RTO	Retransmission TimeOut
RTT	Round-Trip Time
SCTP	Stream Control Transport Protocol TCP

SM	Single-mesh
SO	Sistema Operacional
`ssthresh`	Slow start threashold
sRTT	Smoothed RTT
TCP	Transmission Control Protocol
wVegas	Weighted Vegas

LISTA DE SÍMBOLOS

α	Fator de agressividade.
α_{total}	Total de pacotes acumulados nas filas dos enlaces.
α_r	Fator de agressividade do sub-fluxo r.
$\bar{\tau}_r$	RTT base, sendo o menor observado nas transmissões do sub-fluxo r.
$\ddot{\tau}_r$	RTT do sub-fluxo r.
δ_r	Diferença entre a taxa esperada e taxa de envio atual do sub-fluxo r.
γ_r	Limiar da partida lenta (`sstresh`) do sub-fluxo r.
$\hat{\tau}_r$	Média do RTT no último ciclo de transmissões do sub-fluxo r.
τs	Razão entre o espaço da janela e o sRTT.
RTO_{max}	Maior tempo de esgotamento de temporizador de retransmissão.
BW_r	Largura da banda do sub-fluxo r.
IP_{hdr}	Comprimento do cabeçalho do protocolo IP.
TCP_{hdr}	Comprimento do cabeçalho do protocolo TCP.
b_r	Número de bytes recentemente reconhecidos em r.
f_r	Pacotes pendentes de confirmação (*in-flight*) do sub-fluxo r.
l_{1r}	Bytes transmitidos com sucesso entre as duas últimas perdas do sub-fluxo r.
l_{2r}	Bytes transmitidos após a última perda do sub-fluxo r.
l_r	Número de bytes transmitidos com sucesso entre as duas últimas perdas do sub-fluxo r.
m_r	Tamanho máximo do segmento transmitido no sub-fluxo r.
n_d	Endereços IP de destino.
n_s	Endereços IP de origem.
q_r	Atraso de fila do sub-fluxo r.

r	Sub-fluxo MPTCP que percorre um caminho ou rota r.
r_{best}	Melhor sub-fluxo.
T_r	Taxa de transmissão atual do sub-fluxo r.
w	Janela de congestionamento.
w_{total}	Capacidade total das janelas de congestionamento dos sub-fluxos.
w_r	Janela de congestionamento do sub-fluxo r.
ws	Espaço da janela de congestionamento.

SUMÁRIO

1 INTRODUÇÃO

Atualmente, *desktops* e dispositivos móveis possibilitam diferentes meios de acesso aos serviços e recursos disponibilizados em rede. *Desktops* possuem interfaces de rede local com e sem fio. *Smartphones* podem transmitir dados por rede *WiFi* ou rede celular. Enquanto os nós de rede já estão equipados para que os sistemas finais estejam multi-conectados, a pilha TCP/IP, por consequência de uma arquitetura que se ossificou ao longo da existência da Internet, permite apenas uma rota padrão para acesso à rede. Tal limitação impede que as aplicações façam uso múltiplo de diferentes interfaces de rede no nó e, consequentemente, realizem entrada/saída concorrente sobre essas interfaces. Para contornar essa limitação, o *MultiPath TCP* (MPTCP) propõe uma extensão do protocolo TCP que disponibiliza um conjunto de mecanismos para que sistemas finais possam agregar capacidades de transmissão de diferentes interfaces de rede existentes no mesmo nó (FORD et al., 2013). Sub-fluxos TCP independentes são estabelecidos pelo MPTCP através de diferentes caminhos fim-a-fim entre cliente e servidor. Além de maior taxa de transmissão pela agregação da capacidade individual dos sub-fluxos, o MPTCP possibilita maior tolerância a falhas. Na indisponibilidade de um sub-fluxo, pode haver um ou mais sub-fluxos ativos para continuar a transmissão.

Nesta dissertação, foram analisados experimentalmente o protocolo MPTCP e identificado seu limite de desempenho em diferentes condições de comunicação fim-a-fim. Particularmente, focamos a investigação em dois mecanismos fundamentais do protocolo que interferem diretamente no desempenho das transmissões em multi-fluxos: o escalonador de pacotes e o gerenciador de caminhos. O primeiro é responsável por quebrar o fluxo de *bytes* vindo da aplicação em segmentos e, conforme alguma política de escalonamento, determinar qual sub-fluxo deve ser utilizado no momento do envio do pacote (FORD et al., 2011). Como padrão, o MPTCP utiliza a escalonador LL (*Lowest Latency*), cujo critério de escolha do sub-fluxo no momento do envio do pacote se baseia no sub-fluxo que possui a menor estimativa de tempo de ida e volta (sRTT). O segundo mecanismo é responsável por detectar e sinalizar a presença de múltiplos endereços IP nos sistemas finais, bem como estabelecer e adicionar sub-fluxos sobre esses endereços em uma conexão MPTCP (FORD et al., 2011). O gerenciador de caminhos padrão é o *Full-mesh*, o qual estabelece uma malha de $n_s \times n_d$ sub-fluxos de acordo com o número de interfaces endereçadas na origem e destino, respetivamente.

1.1 Motivação

O MPTCP é um protocolo de propósito geral, com aplicações diversas (BONAVENTURE; PAASCH; DETAL, 2017). Em *datacenters*, a vazão de dados pode ser melhorada com transmissão sobre diversos caminhos entre servidores (RAICIU et al., 2011). Dispositivos mó-

veis que possuem duas interfaces de comunicação sem fio, como 4G e WiFi, podem se beneficiar com MPTCP com a transmissão concorrente sobre as interfaces e maior tolerância a falhas de conexão durante a mobilidade dos usuários. O MPTCP tem sido desenvolvido e implantado em diversos sistemas operacionais (BONAVENTURE; PAASCH; DETAL, 2017), como Linux, iOS Apple, Citrix, FreeBSD, Oracle Solaris. A maior implantação do protocolo tem sido em dispositivos móveis, possivelmente com centenas de milhões de dispositivos habilitados atualmente. Desde 2013, dispositivos com iOS7 da Apple implementam MPTCP, onde multi-fluxos são utilizados para transmissão de dados gerados pela aplicação *Siri*, de reconhecimento/controle de voz, os quais demandam grande processamento de servidores em *datacenter* (BONAVENTURE; SEO, 2016).

O desempenho do MPTCP é determinado pela cooperação de três principais mecanismos: escalonamento de pacotes, controle de congestionamento e gerenciamento de caminhos. Entretanto, as aplicações são diversas e os possíveis fim-a-fim entre as aplicações possuem características distintas, de como latência, taxa de erro e capacidade de transmissão. Melhorar o desempenho do protocolo, considerando a ampla diversidade e heterogeneidade dos caminhos fim-a-fim, é um desafio aberto sendo atacado por diversos pesquisadores através de algoritmos mais eficientes para cada um desses mecanismos.

A necessidade de escalonamento em sistemas computacionais foi inicialmente identificada no compartilhamento de recursos em sistemas de *time-sharing* (LIU; LAYLAND, 1973), por exemplo, uma única CPU para vários processos. No contexto de transmissão de dados em redes de computadores, escalonamento se torna necessário em sistemas multi-conectados, onde é preciso tomar decisões sobre qual dispositivo de entrada/saída será utilizado para transmitir um fluxo de pacotes ou um pacote individualmente. Embora escalonamento não seja um problema recente em sistemas computacionais, é um problema recorrente, dado o surgimento de novas tecnologias de comunicação, como o protocolo MPTCP, proposto nos últimos cinco anos (FORD et al., 2011). No contexto do MPTCP, uma conexão de múltiplos caminhos $\mathcal{R}$ é formada diversos sub-fluxos r, que operam de forma semelhante as conexões TCP ordinárias. Cabe ao escalonador do MPTCP determinar o melhor $r_{best} \in \mathcal{R}$ no momento que antecede o envio de um pacote. Entretanto, o escalonador atualmente existente no protocolo utiliza apenas um critério de decisão, baseado na latência do sub-fluxo, independente de outras variáveis de determinam as condições dos caminhos, como capacidade da transmissão e taxa de perda de pacotes. Portanto, o problema de escalonamento no MPTCP ainda é um campo a ser explorado, tendo espaço para investigação e proposição de novas soluções cientes do contexto das condições de rede em que se encontram os possíveis caminhos entre dois sistemas finais.

Assumindo que um sub-fluxo MPTCP seja um recurso de transmissão de dados, pode-se, à primeira vista, assumir a premissa de que quanto mais sub-fluxos houver em uma conexão $\mathcal{R}$, maior a quantidade de recursos disponíveis para transmissão e, portanto, maior será o desempenho da conexão. Especificamente no contexto do MPTCP, não se pode assumir tal

premissa. Ao competirem pelo uso dos gargalos de cada um dos possíveis caminhos fim-a-fim existentes entre dois sistemas finais multi-conectados, os sub-fluxos $r \in \mathcal{R}$ são controlados por um algoritmo de controle acoplado de congestionamento. Nesse caso, não necessariamente mais sub-fluxos implica em maior desempenho. O mecanismo responsável por estabelecer os sub-fluxos e, portanto, definir a sua quantidade em uma conexão MPTCP é o gerenciador de caminhos. Com a finalidade de encontrar caminhos disponíveis entre dois sistemas finais, o gerenciador de caminhos pode estabelecer sub-fluxos que compartilham de um mesmo gargalo, mesmo em caminhos disjuntos, gerando concorrência pelo seu uso entre os próprios sub-fluxos.

A decisão de escalonamento do sub-fluxo no ato do envio do pacote pode levar ao aumento ou então à degradação da vazão de dados fim-a-fim. Da mesma forma, o método de estabelecimento de sub-fluxos utilizado pelo gerenciador de caminhos impacta no desempenho da conexão MPTCP. A partir dos primeiros experimentos realizados, identificamos problemas de desempenho nas soluções padrão do MPTCP. A Figura 1 apresenta o desempenho obtido em experimentos iniciais com escalonador LL e gerenciador *Full-mesh* com transmissões multi-fluxos sobre 5 caminhos de capacidade heterogêneas. Os detalhes dos experimentos realizados são discutidos no Capítulo 5.

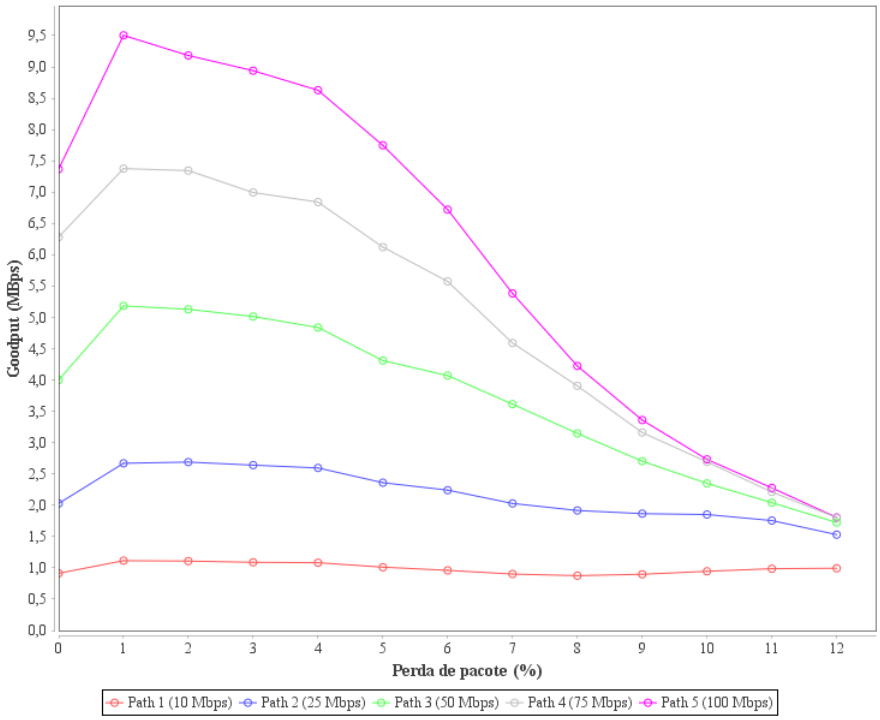


Figura 1 – Desempenho obtido em transmissão multi-fluxo por caminhos heterogêneos sob escalonador LL e gerenciador *Full-mesh*.

Os problemas identificados decorrentes da operação do escalonador LL e o gerenciador *Full-mesh* afetam o desempenho das transmissões em multi-fluxos, reduzindo a vazão de dados em até 2 vezes, dependendo da situação encontrada no cenário. Esses problemas são discutidos adiante.

1.1.1 Degradação de desempenho na decisão de escalonamento

O escalonador LL, ao priorizar sub-fluxos de baixa latência através do menor sRTT, pode tomar decisões inconsistentes quando os caminhos estão congestionados, sob frequente perda de pacotes e redução da janela de congestionamento. O impacto das decisões pode ser observado no gráfico da Figura 1. Nota-se, pela vazão resultante, que os sub-fluxos tendem a não ser diferenciados conforme o aumento da taxa de perda de pacotes do caminho, mesmo quando os caminhos possuem capacidades bem distintas de transmissão. Nesse caso, o sub-fluxo de menor latência, mas com maior taxa de perda de pacotes, não resulta em uma boa decisão de escalonamento do ponto de vista de desempenho. Da mesma forma, escalonar um sub-fluxo rápido, de baixa estimativa sRTT, mas de baixa taxa de transmissão não irá aumentar a vazão do multi-fluxo de dados.

1.1.2 Degradação de desempenho no gerenciamento de caminhos

A malha de $n_s \times n_d$ sub-fluxos estabelecidos em *Full-mesh* causa degradação de desempenho da conexão MPTCP sobre caminhos que não estão congestionados, sem perda de pacotes. No gerenciador padrão, para cada par de endereços s_i e d_j disponíveis na origem (cliente) e destino (servidor), respectivamente, é estabelecido um sub-fluxo r_{ij}. Portanto, se o número de interfaces endereçáveis n_s e/ou n_d for grande, o número de sub-fluxos estabelecidos também será. Isso implica em ter $\max(n_s, n_d)$ sub-fluxos compartilhando um mesmo enlace de gargalo de cada caminho fim-a-fim. O impacto desse gerenciamento pode ser observado no gráfico da Figura 1. Observa-se que em caminhos que não geram perdas de pacote (taxa de 0 %), o congestionamento *inter-path* de $\max(n_s, n_d)$ sub-fluxos é agressivo ao ponto de a vazão agregada dos multi-fluxos ser inferior a de uma conexão TCP de caminho único.

1.2 Objetivos

Neste trabalho, investigamos experimentalmente soluções alternativas, de escalonamento de pacotes e de gerenciamento de caminhos, com o objetivo de melhorar o desempenho de transmissões em multi-fluxos MPTCP e, consequentemente, aprimorar o protocolo. Para tanto, objetivos mais específicos de pesquisa foram definidos:

- Compreender o Estado da Arte no que se refere a escalonamento de pacotes e gerenciamento de caminhos.
- Definir e implantar um ambiente em laboratório para experimentação de transmissões de multi-fluxos MPTCP.
- Investigar experimentalmente possíveis problemas que resultam em degradação de desempenho de transmissões de multi-fluxos MPTCP.

- Propor, implementar, avaliar e comparar soluções alternativas para os problemas identificados.

- Discutir em maior profundidade resultados experimentais das soluções propostas.

1.3 Metodologia

Para realizar a investigação pretendida, utilizamos a abordagem de planejamento de experimentos discutida em Paasch, Khalili e Bonaventure (2013) e selecionamos os principais fatores que influenciam o desempenho do MPTCP. Tais fatores foram mapeados em parâmetros de comunicação fim-a-fim, a saber: taxa da transmissão, taxa de perda de pacotes e atraso. Para cada fator, determinamos o domínio específico de possíveis valores de parametrização de acordo com estudos da Literatura sobre medição de transmissão de dados na Internet fixa e móvel. Assim, condições bem realistas de caminhos fim-a-fim, de péssimas a excelentes, foram caracterizadas, possibilitando a experimentação de cenários com ampla diversidade de multi-fluxos MPTCP, os quais podem ocorrer em transmissões reais na Internet.

Assim como em Paasch et al. (2014a) e Paasch et al. (2014b), implantamos uma ambiente experimental ideal através de uma topologia de rede mínima, onde duas máquinas, cliente e servidor, foram conectadas ponto-a-ponto. Tal topologia, ainda que minimalista, permitiu:

a) criar um ambiente de experimentação totalmente controlado com flexibilidade para configurar os fatores selecionados e os domínios definidos;

b) realizar a avaliação e identificação do desempenho máximo que soluções MPTCP podem obter em diferentes condições de comunicação fim-a-fim na Internet.

1.4 Resultados Obtidos

Os principais resultados concretos obtidos com a pesquisa realizada neste trabalho foram:

i) Três novos escalonadores de pacotes: *Largest Window Space* (LWS), *Lowest Time/Space* (LTS) e o *Highest Sending Rate* (HSR). O escalonador LWS utiliza o indicador de espaço de dados existente na janela de congestionamento para determinar qual sub-fluxo será escolhido. Em sua estratégia, LWS seleciona o sub-fluxo de maior espaço entre os sub-fluxos estabelecidos na conexão MPTCP. LTS, de forma semelhante, utiliza como indicador a razão entre a estimativa de tempo de ida e volta e o espaço na janela, escolhendo o sub-fluxo de menor relação contida nesse indicador. Já HSR, utiliza o indicar de taxa de transmissão instantânea, escolhendo o sub-fluxo que possui a maior taxa atual.

ii) Um novo gerenciador de caminhos, denominado *Single-mesh*. Tal gerenciador limita o estabelecimento de sub-fluxos em $min(n_s, n_d)$, sendo apenas um sub-fluxo por interface endereçada no lado cliente da comunicação. Ao limitar os sub-fluxos na conexão MPTCP dessa forma, *Single-mesh* é capaz de mitigar o congestionamento *inter-path* verificado em *Full-mesh*, garantindo que haja no máximo um sub-fluxo MPTCP por caminho fim-a-fim, ou seja, apenas um sub-fluxo MPTCP por enlace de gargalo em cada caminho fim-a-fim.

ii) Definição de um ambiente experimental mínimo em laboratório através de apenas duas máquinas conectadas ponto-a-ponto. Tal ambiente consiste de poucos componentes de *hardware*[1] e vários componentes de *software open-source*[2]. Embora mínimo, o ambiente é totalmente controlável na configuração dos *softwares* de rede envolvidos, sendo capaz de prover flexibilidade na execução de experimentos que exigem elevada diversidade de transmissões MPTCP e, ao mesmo tempo, livre de ruídos não intencionados.

A partir dos experimentos realizados, observamos que as soluções propostas de escalonamento e gerenciamento de caminhos possibilitaram maior desempenho que as soluções padrão do MPTCP. Tal desempenho foi observado através do aumento da vazão fim-a-fim em cenários diversos, sob diferentes condições de comunicação fim-a-fim, e com uso dos mecanismos existentes de controle de congestionamento do MPTCP, como LIA (RAICIU; HANDLEY; WISCHIK, 2011), OLIA (KHALILI et al., 2013), BALIA (WALID et al., 2015) e wVegas (CAO; XU; FU, 2016).

Nos experimentos com caminhos de capacidade heterogênea e alta taxa de perda de pacote, o escalonador LWS obteve aproximadamente 7 % mais vazão que LL nos controles de congestionamento LIA, OLIA e wVegas. Em BALIA, contudo, o ganho foi de apenas 1 %. LTS teve desempenho pouco inferior ao LWS, obtendo aproximadamente 4 % mais vazão que LL com uso dos controles de congestionamento LIA, OLIA e wVegas. Nos experimentos com caminhos de características heterogêneas (em taxas de perda de pacotes, taxas de transmissão e latências), os escalonadores HSR e LTS obtiveram ganhos gerais de aproximadamente 2 % sobre o LL, com ganhos mais expressivos nos cenários específicos de cada experimento. Nos resultados obtidos com *Single-mesh*, o gerenciador de caminhos mostrou ser eficiente para os controles de congestionamento OLIA e BALIA, obtendo mais que o dobro da vazão de *Full-Mesh* sobre caminhos não congestionados de capacidades heterogêneas.

1.5 Organização do Texto

O restante desta dissertação está organizado em outros cinco capítulos. No Capítulo 2 apresentamos maiores detalhes sobre os protocolos TCP e MPTCP, com destaque para os meca-

[1] Duas máquinas, duas placas de rede de múltiplas portas e cabos de rede.

[2] Distribuição Linux com a implementação oficial do protocolo MPTCP, além de diversas ferramentas: codificação; parametrização dos caminhos sobre a rede; geração, captura e análise de tráfego.

nismos e operações essenciais de transporte de dados sobre fluxos de caminhos únicos e fluxos de múltiplos caminhos, respetivamente.

No Capítulo 3 discutimos os trabalhos relacionados à nossa investigação, abordando os algoritmos de controle de congestionamento, os escalonadores de pacotes e os gerenciadores de caminhos existentes no MPTCP, além dos principais trabalhos propostos na Literatura.

No Capítulo 4 apresentamos e discutimos detalhes das soluções que propomos através dos escalonadores LWS, LTS e HSR e do gerenciador de caminho *Single-mesh*.

No Capítulo 5 apresentados detalhes do ambiente experimental e da metodologia que adotamos para o planejamento dos experimentos. Provemos uma análise de desempenho, com discussão dos resultados obtidos em diversos experimentos com cada solução proposta, utilizando como base de comparação as soluções padrão do MPTCP, como por exemplo. LL e *Full-mesh*.

Por fim, no Capítulo 6 trazemos as principais conclusões deste trabalho e um direcionamento para investigações futuras.

2 PROTOCOLOS FUNDAMENTAIS DE TRANSPORTE DE DADOS

Iniciamos este capítulo com a definição dos principais conceitos relacionados à transmissão por fluxos e multi-fluxos fim-a-fim. Serão apresentados a descrição de fundamentos dos protocolos TCP e MPTCP, que são utilizados nessas transmissões.

2.1 Principais Conceitos

Na Literatura encontramos definições de conceitos relacionados à transmissão por fluxos e multi-fluxos fim-a-fim:

Definição 2.1 (*Multi-home*) *Capacidade de um sistema final habilitar vários recursos de transmissão. Um sistema final é dito ser multihomed se o mesmo possuir múltiplas interfaces de redes ativas, habilitando diversas rotas de acesso à rede através de vários endereços de rede IP (IYENGAR; AMER; STEWART, 2006).*

Definição 2.2 (*Path*) *Um caminho (path) consiste em uma sequência de enlaces existentes entre dois sistemas finais, emissor e o receptor. No contexto do MPTCP, um caminho é definido e identificado pelo sistema final através de uma tupla de quatro elementos: pares de endereços e portas de origem e destino (FORD et al., 2013).*

Definição 2.3 (*Multi-path*) *Existência de mais de um caminho em sistemas finais multihomed, onde protocolos multi-path, como o MPTCP, habilitam transporte concorrente sobre os múltiplos caminhos (Concurrent Multipath Transport - CMT), melhorando a utilização dos recursos de rede e o desempenho de transmissão com um aumento da vazão, além de prover maior tolerância a falhas na transmissão de dados (IYENGAR; AMER; STEWART, 2006).*

Definição 2.4 (Sub-fluxo MPTCP) *Fluxo de segmentos TCP operando sobre um caminho individual que, por sua vez, é parte da formação de uma conexão maior no MPTCP. Um sub-fluxo é iniciado e terminado similarmente a uma conexão TCP comum (FORD et al., 2013).*

Definição 2.5 (Conexão TCP) *Conexão lógica estabelecida sobre um endereço origem e um destino entre dois sistemas finais, permitindo trafego confiável de dados sobre um caminho não confiável entre eles. Uma conexão TCP é definida por um par de sockets nos sistemas finais, que identifica os dois lados da conexão (POSTEL et al., 1981).*

2.2 Transmission Control Protocol (TCP)

TCP é o principal protocolo de transporte de dados para sistemas finais na Internet. Tal protocolo provê mecanismos para realizar transmissões confiáveis entre cliente e servidor, garantindo que dados cheguem ao destino, mesmo sobre enlaces não confiáveis, sujeitos à perda de pacotes (POSTEL et al., 1981).

2.2.1 Estabelecimento e Encerramento de Conexões

O estabelecimento de uma conexão TCP é executado por procedimento em três vias (*three-way handshake*). Na primeira etapa, o sistema final cliente, que é a entidade ativa que deseja estabelecer a conexão, envia para o sistema final servidor o primeiro pacote controle com a *flag* de sincronização (`SYN`) ativa. Quando o servidor de destino recebe o pacote `SYN`, o mesmo responde com outro pacote de controle `SYN` que, desta vez, contém também a *flag* `ACK` ativa para sinalizar o reconhecimento do pacote `SYN` enviado pelo cliente. Por fim, ao receber o pacote `SYN/ACK` do servidor, o cliente envia um último pacote de controle `ACK` confirmando o estabelecimento da conexão (POSTEL et al., 1981). A Figura 2 ilustra o estabelecimento da conexão TCP.

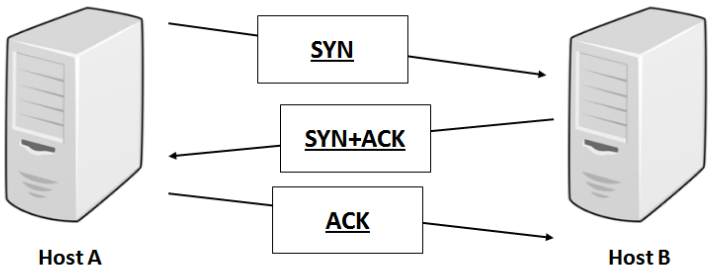


Figura 2 – Estabelecimento de uma conexão TCP (POSTEL et al., 1981).

A operação de encerramento da conexão deve ser realizada por ambas as partes (POSTEL et al., 1981). A ação de fechamento se inicia quando um sistema final envia um pacote com as *flags* `FIN` e `ACK` ativas. Quando o sistema final de destino recebe essa notificação de encerramento, o mesmo responde com um pacote `ACK`, assim entrando em processo de encerramento. O sistema final de destino aguarda que todos os pacotes pendentes sejam confirmados e então envia também o seu pacote `FIN/ACK` para o sistema final que iniciou o encerramento. Ao receber a notificação da outra parte, o sistema final iniciante responde apenas com um pacote `ACK`. A Figura 3 ilustra essa operação.

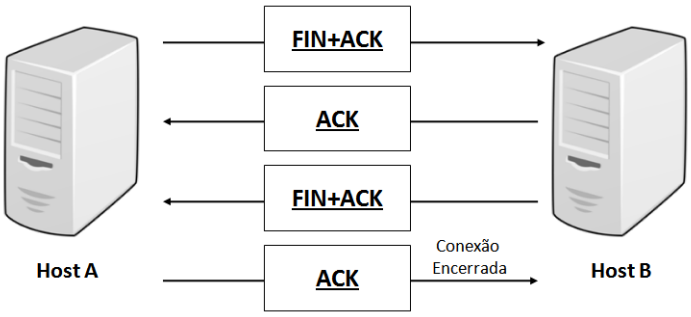


Figura 3 – Encerramento de uma conexão TCP (POSTEL et al., 1981).

2.2.2 Mecanismos de Confiabilidade

Para prover confiabilidade na transmissão de pacotes sobre de meios físicos não confiáveis, o protocolo TCP possui cinco principais mecanismos:

- **Número de Sequência**. Cada pacote enviado é marcado com um número de sequência pelo emissor. Esse número é definido pela quantidade de bytes que está sendo encapsulado no pacote TCP, portanto, esse número é incremental a cada envio. Isso permite que o emissor identifique unicamente o pacote em transmissão, sendo possível determinar a retransmissão de um pacote específico que foi perdido nos envios. No lado receptor, o número de sequência permite ordenar dados recebidos, mesmo que os pacotes cheguem desordenados, validando todo recebimento (POSTEL et al., 1981).

- **Retro-alimentação do Destinatário**. A garantia da chegada dos pacotes no sistema final de destino ocorre por meio de confirmações de recebimento com pacotes `ACK`. Esses pacotes notificam o emissor com um valor de número de reconhecimento $i + 1$, o que é utilizado para: confirmar que o pacote de número de sequencia i foi recebido sem erro e ordenado pelo receptor; indicar o número de sequência $i + 1$ do próximo pacote a ser enviado pelo emissor.

- **Soma de Verificação**. Devido a ruídos no meio de transmissão, um pacote pode sofrer alteração em seu conteúdo, gerando erro de pacote no recebimento. O TCP utiliza somas de verificação (*checksum*) no pacote como mecanismo para determinar se o mesmo está ou não corrompido. Esse mecanismo calcula o valor de *checksum* de todo pacote a ser enviado e o inclui em campo específico do cabeçalho TCP. Tão logo o pacote é recebido, o valor de *cheksum* é localmente calculado pelo receptor, que compara esse valor recém calculado com o valor *checksum* que veio no cabeçalho do pacote. Caso haja divergência entre os valores, o receptor infere que o pacote está corrompido e o descarta, garantindo transmissão sem erros de pacotes (POSTEL et al., 1981).

- **Temporizador de Retransmissão**. Um pacote transmitido por um meio não confiável pode ser perdido nos saltos ao longo do caminho fim-a-fim e não chegar no destino. Como o nó de origem aguarda uma resposta da confirmação de recebimento do nó de destino, um temporizador de retransmissão RTO (*retransmission timeout*) é utilizado no emissor para aguardar um tempo máximo para o recebimento do pacote `ACK`. Caso o temporizador esgote, o emissor então reenvia o pacote pendente de `ACK`. O temporizador de retransmissão é baseado no tempo RTT (*round -trip time*) de ida e volta do pacote, ou seja, é o tempo que leva para o emissor transmitir o pacote de sequência i e receber o pacote de confirmação $i+1$ do receptor. Para determinar o temporizador RTO, o TCP emissor primeiro mede o RTT do pacote i e, então, estima o tempo de ida e volta (sRTT) e sua variação (RTTvar) para o próximo pacote $i+1$. As estimativas são determinadas por média móvel exponencial ponderada (PAXSON; ALLMAN, 2000):

$$\text{sRTT}_{i+1} = (1-\alpha) \times \text{sRTT}_i + \alpha \times \text{RTT}_i, \tag{2.1}$$

$$\text{RTTvar}_{i+1} = (1-\beta) \times \text{RTTvar}_i + \beta \times |\text{sRTT}_i - \text{RTT}_i|, \tag{2.2}$$

  onde: α e β são pesos que, segundo Jacobson (1995), possuem os valores padrão de $1/8$ e $1/4$, respectivamente; e sRTT_i, RTT_i e RTTvar_i são a i-ésima estimativa, medição e desvio conhecidos do tempo de ida e volta. Após estimar sRTT e RTTvar, o emissor determina o temporizador de retransmissão RTO para o próximo pacote a ser enviado da seguinte forma (ALLMAN; PAXSON, 1999):

$$\text{RTO}_{i+1} = \text{sRTT}_{i+1} + k \times \text{RTTvar}_{i+1} \tag{2.3}$$

  onde k é fator arbitrário, cujo valor padrão é 4.

- **Recuperação Rápida**. Se múltiplos pacotes são enviados de acordo com o tamanho atual da janela de congestionamento w, caso haja perda, o receptor irá confirmar pacotes de forma cumulativa apenas com o número de sequência do último pacote que chegou em ordem. O emissor então irá receber pacotes duplicados de reconhecimento `ACK` à medida que pacotes cheguem fora de ordem no receptor. O mecanismo de recuperação rápida (POSTEL et al., 1981) prevê que o emissor assuma a perda do pacote tão logo receber três pacotes `ACK` duplicados com o número de sequência do último pacote que chegou ordenado no destino. Isso evita que o emissor dependa do esgotamento do temporizador RTO, o que, em geral, demandaria em um tempo maior para detectar a perda. Essa mesma operação é realizada para o pacote que é recebido com erro no destino. Nesse caso, o receptor também confirmará com um pacote `ACK` o número de sequência do último pacote que chegou corretamente e em ordem.

2.2.3 Controle de Congestionamento

Além de mecanismos de confiabilidade, o TCP provê também controle de congestionamento, cuja função é limitar a taxa de *bytes* a serem transmitidos sobre uma conexão fim-a-fim,

conforme a percepção de congestionamento na rede. A taxa de envio é determinada por w/RTT *bytes* por segundo, onde w é o comprimento atual da janela de congestionamento (`cwnd`). A janela de congestionamento então indica a quantidade de *bytes* de dados que o emissor poderá enviar a cada a instante de envio de pacotes.

A percepção de congestionamento na rede é comummente determinada pela perda de pacote. Essa condição é detectada através do esgotamento de temporizador de RTO ou por recebimento de pacotes de controle `ACK` duplicados no emissor, como introduzido na sub-seção anterior. O modo como a janela w é retraída ou expandida depende do algoritmo de controle de congestionamento. O algoritmo mais utilizado hoje é a versão chamada *NewReno* (ALLMAN; PAXSON; BLANTON, 2009), o qual prevê duas fases de controle de congestionamento durante o tempo de vida de uma conexão TCP:

i) **Partida Lenta**. Essa fase, chamada de *slow start*, acontece no início de uma conexão TCP. O valor inicial de janela w é definido por 1 MSS (*Maximum Segment Size*), sendo

$$\text{MSS} = \text{MTU} - |\text{TCP}_{hdr}| - |\text{IP}_{hdr}|, \tag{2.4}$$

onde, tipicamente, a unidade máxima de transmissão MTU (*Maximum Transmission Unit*) de quadro de dados é de 1500 bytes na Internet e o comprimento do cabeçalho de cada protocolo é $|\text{TCP}_{hdr}| = 20$ bytes e $|\text{IP}_{hdr}| = 20$ bytes. Isso corresponde a um MSS comumente limitado a 1460 bytes de dados da aplicação (*payload*) que um pacote TCP pode transportar até o destino. Após a confirmação do recebimento do primeiro pacote enviado, o comprimento da janela w é aumentado em 1 MSS e, então, o emissor pode enviar de uma vez mais dois segmentos na próxima transmissão. O tamanho de w cresce em 1 MSS a cada confirmação de recebimento de pacote pelo destinatário. Esse aumento tem comportamento exponencial e ocorre até que o temporizador de envio se esgote devido a perda de um pacote não confirmado. Essa perda é utilizada pelo emissor como percepção de um possível congestionamento no caminho fim-a-fim entre os sistemas finais de origem e destino. Então, para evitar que o caminho seja mais sobrecarregado e, consequentemente, ocorra mais perdas, a transmissão é retomada a partir de um limiar de janela, chamado de `ssthresh` (*slow start threshold*), determinado por $w/2$. Quando a janela w é reduzida para o valor de `ssthresh`, inicia-se uma nova fase chamada de prevenção de congestionamento.

ii) **Prevenção de Congestionamento**. Após a fase de partida lenta, a janela w é definida pela metade do comprimento da janela expandida até o evento da perda de pacote. Neste caso, após perceber o congestionamento, o emissor não aumentará de forma exponencial a janela w conforme as confirmações recebidas do destinatário. De forma mais prudente, na fase de prevenção de congestionamento (*congestion avoidance*), o crescimento da janela já se apresenta de forma linear, incrementando a janela a uma fração do MSS a cada confirmação de pacote recebido pelo destino. O estado de prevenção de congestionamento

é atualizado no emissor também pelo evento de perda de pacote. Nesse caso, um novo valor para `ssthresh` é definido, sendo a metade da janela w atual.

Sob o evento de perda, o valor da janela w é reduzido pela metade, conforme `ssthresh`, reiniciando uma nova fase de prevenção de congestionamento, onde o crescimento da janela irá ocorrer de forma linear. A expansão linear e retração drástica da janela é conhecida na Literatura como crescimento aditivo e diminuição multiplicativa AIMD (*additive increase, multiplicative decrease*).

2.3 MultiPath TCP (MPTCP)

Multipath TCP (MPTCP) é composto por um conjunto de extensões do TCP (*Transmission Control Protocol*) para permitir o transporte de pacotes sobre múltiplos caminhos (*paths*) simultaneamente entre sistemas finais de origem e destino. O uso concorrente de caminhos múltiplos naturalmente aumenta a taxa de transmissão entre os sistemas finais envolvidos, bem como provê maior tolerância a falhas de comunicação a partir da redundância de múltiplos sub-fluxos TCP existentes (FORD et al., 2013; FORD et al., 2011).

2.3.1 Caminhos Múltiplos

Ao estender o protocolo TCP na Camada de Transporte, o MPTCP é transparente aos protocolos das camadas adjacentes, de Aplicação e Rede. Assim, não é necessário nenhuma mudança no código das aplicações TCP para realizar sobre MPTCP. Para possibilitar transmissão com múltiplos caminhos entre dois sistemas finais, o MPTCP gerencia a operação de um conjunto de sub-fluxos TCP, como ilustrado na Figura 4. Um sub-fluxo é, do ponto de vista operacional da rede, uma conexão TCP independente que, contudo, está associada a uma conexão maior MPTCP.

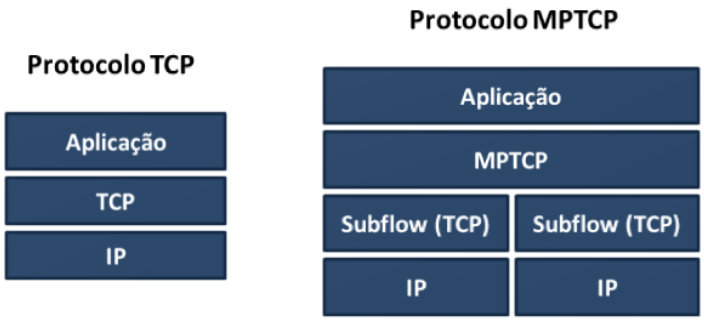


Figura 4 – Comparação entre as pilhas TCP e MPTCP. Fonte: imagem adaptada de (FORD et al., 2013).

A Figura 5 ilustra em alto nível uma transmissão envolvendo dois sistemas finais MPTCP, cada nó possuindo duas interfaces de rede. Portanto, cada um pode estar ligado a duas redes físicas e lógicas distintas. Uma conexão MPTCP é estabelecida a partir de uma conexão TCP entre os nós A e B através dos seus endereços A1 e B1. Após ser estabelecida a conexão MPTCP, caso seja identificada a possibilidade de caminhos adicionais a partir de outros endereços IP existentes nos nós, são estabelecidas conexões TCP adicionais (sub-fluxos) sobre cada caminho. Os sub-fluxos fazem parte de uma única conexão MPTCP. Esse cenário é ilustrado na figura com a conexão estabelecida entre o endereço A2 e o endereço B2.

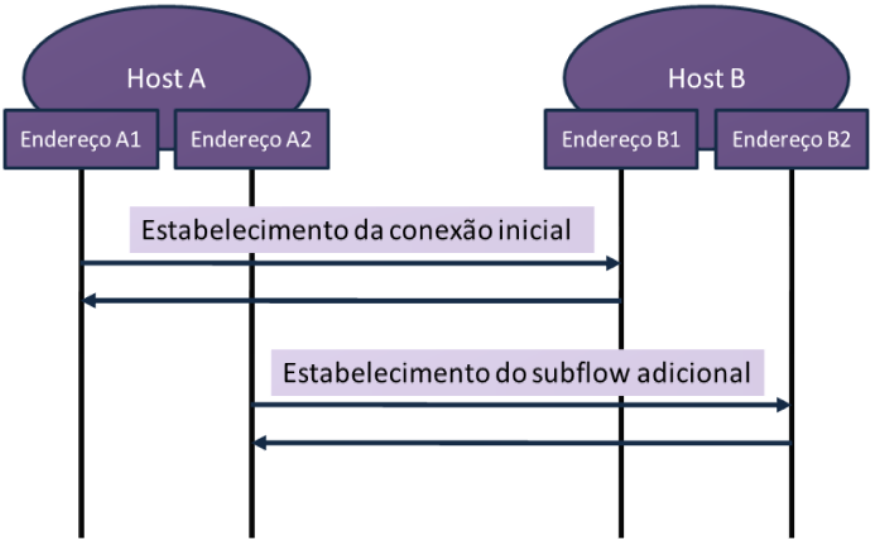


Figura 5 – Estabelecimento inicial da conexão MPTCP e adição de um novo sub-fluxo. Fonte: imagem adaptada de (FORD et al., 2013).

Novos caminhos são identificados quando há vários endereços IPs nos sistemas finais. As descobertas e configurações dos sub-fluxos são realizadas através de um gerenciador de caminhos. O MPTCP utiliza números de sequência globais, como se fossem no nível de uma única conexão, de modo a possibilitar a remontagem dos segmentos que chegam dos vários sub-fluxos. Cada sub-fluxo, por sua vez, utiliza seu espaço local de número de sequência como no TCP. Os sub-fluxos são terminados como nas conexões TCP, através da finalização em quatro vias, a partir de pacotes de controle `FIN`. Já a conexão MPTCP é terminada ao enviar um pacote `FIN` de nível global.

O protocolo MPTCP gerencia um conjunto de sub-fluxos, sendo responsável pela divisão e junção dos dados que são enviados e recebidos sobre sub-fluxos. O número de sub-fluxos não é fixo durante a existência de uma conexão MPTCP, de modo que sub-fluxos podem ser removidos ou adicionados à medida que falhas sejam detectadas ou novos caminhos forem descobertos pelo gerenciador do protocolo.

2.3.2 Estabelecimento da Conexão MPTCP

Uma conexão é criada a partir do estabelecimento do primeiro sub-fluxo MPTCP, normalmente através do estabelecimento de uma conexão TCP realizado em três vias (*three-way handshake*) com os pacote de controle `SYN`, `SYN/ACK` e `ACK`. Contudo, há uma pequena diferença. No estabelecimento da conexão, os pacotes carregam a opção `MP_CAPABLE`, sinalizando à outra parte que o mesmo possui suporte para o protocolo MPTCP, conforme ilustra a Figura 6.

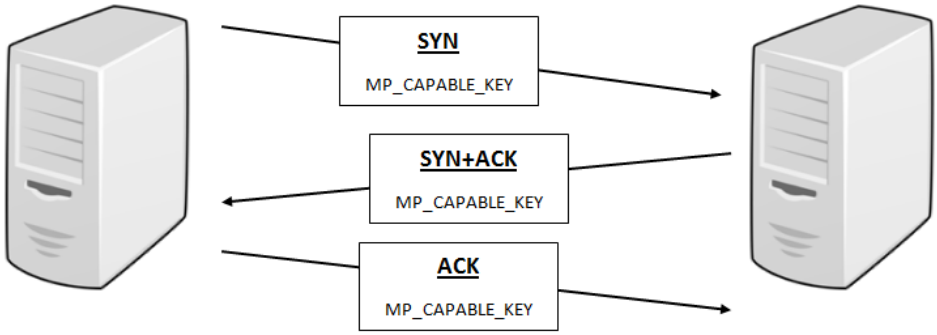


Figura 6 – Estabelecimento do primeiro sub-fluxo. Imagem adaptada de (PAASCH; BONAVENTURE et al., 2014).

O primeiro sub-fluxo estabelecido é denominado sub-fluxo mestre. Através dele, com a opção `MP_CAPABLE`, os sistemas finais trocam material de autenticação (*key*) utilizado principalmente no estabelecimento de novos sub-fluxos e remoção de sub-fluxos existentes. A autenticação dos sub-fluxos é necessária para prover segurança entre as partes envolvidas, uma vez que sub-fluxos podem ser adicionados e/ou removidos ao longo da conexão MPTCP, podendo sofrer ataques de personificação, homem-do-meio e sequestro de sessão. As chaves trocadas na opção `MP_CAPABLE` são utilizadas em autenticações futuras, quando novos sub-fluxos forem estabelecidos.

2.3.3 Adição e Remoção de sub-fluxos

Os endereços IP associados às múltiplas interfaces de um dispositivo podem variar ao longo de uma conexão MPTCP, especialmente em dispositivos móveis, que podem ter interfaces re-endereçadas conforme são migradas em diferentes redes de acesso. Caso haja um novo endereço disponível para uso no nó após o estabelecimento do sub-fluxo mestre, outros sub-fluxos podem ser estabelecidos através desse novo caminho. De forma implícita, mesmo que o par remoto não tenha conhecimento desse novo caminho do par local, quando o sub-fluxo for estabelecido, o par remoto estará ciente que o novo endereço da outra parte pertence a uma conexão MPTCP existente. De forma explícita, novos endereços podem ser anunciados ao par

remoto através de um pacote de controle com a opção `ADD_ADDR` contendo o novo endereço IP e o ID.

Para adicionar sub-fluxos em uma conexão MPTCP em andamento, os pacotes de estabelecimento de conexão `SYN`, `SYN/ACK` e `ACK` carregam a opção `MP_JOIN`. Nesse caso, o par local gera um *token* a partir da chave conhecida do par remoto, a qual foi trocada no estabelecimento do sub-fluxo mestre. O *token*, um número aleatório NONCE (*number once used*) e um endereço `ID` de base para identificação são encapsulados na opção `MP_JOIN`. Assim, quando um sistema final recebe um pacote `MP_JOIN`, o mesmo terá dados para identificar qual conexão MPTCP pertence o sub-fluxo que está sendo estabelecido. Com a chave conhecida do emissor e o número `NONCE` recebido no pacote, é calculado o código HMAC (*Hash-based Message Authentication Code*) para a autenticação de mensagem baseada em *hash*. Esse código é enviado junto ao pacote `MP_JOIN` como material de autenticação. Após autenticação e trocas de endereços ID durante o *three-way handshake*, o novo *subflow* é estabelecido.

O endereço ID atua como um identificador único da conexão, não sendo manipulado em traduções de endereço de rede NAT (*Network Address Translation*). Esse endereço é usado pelo MPTCP para identificar o endereço da origem, de modo que, mesmo que um endereço IP da origem seja alterado durante o seu percurso por um equipamento NAT, com o endereço ID é possível identificar qual sessão o respectivo sub-fluxo pertence.

De forma semelhante, a remoção dos sub-fluxos são realizados através de um envio de pacote de controle com a opção `REMOVE_ADDR`, contendo o endereço ID indicando a operação de remoção. A ação remove todos os sub-fluxos que estavam estabelecidos e utiliza o endereço informado.

2.3.4 Encerramento de Conexão

Para finalizar uma conexão MPTCP, a opção `DATA_SEQUENCE_SIGNAL` deve conter uma indicação de *flag* `FIN`, sinalizando que a sessão será finalizada. A finalização ocorre da mesma forma como no protocolo TCP, mas no nível global de conexão MPTCP. Assim, quando todos os dados forem recebidos com sucesso, um pacote de confirmação é enviado no nível de sessão com um `DATA_ACK`.

2.3.5 Controle de Fluxo

Todos os sub-fluxos $r \in \mathcal{R}$ compartilham o mesmo *buffer* de recepção e informam uma mesma janela de recepção. O tamanho do *buffer* de recepção, para não afetar os outros sub-fluxos em caso de perda ou falha na transmissão, é determinado por

$$\left(\sum_{r \in \mathcal{R}} \mathrm{BW}_r \right) \times \mathrm{RTO}_{max}, \tag{2.5}$$

onde BW_r é a largura de banda de cada sub-fluxo r e RTO_{max} é o maior tempo de esgotamento de temporizador de retransmissão existente entre os sub-fluxos.

Como o *buffer* é compartilhado por vários sub-fluxos, cada um com uma condição de transmissão, o seu tamanho deve ser definido para que os sub-fluxos não sofram suspensão na transmissão por falta de espaço no *buffer*. Tendo em vista que a sensibilidade entre os diversos sub-fluxos seria algo complexo de ser determinado (FORD et al., 2011), a magnitude tanto do *buffer* de envio quanto do *buffer* de recebimento pode ser determinada de forma mais simplificada:

$$\text{buffer} = 2 \times \sum (\text{BW}_r) \times \text{RTT}_{max} \tag{2.6}$$

onde é RTT_{max} é o maior RTT entre todos os sub-fluxos.

As confirmações no MPTCP são tratadas nos sub-fluxos, normalmente com os números de sequência `ACKs` no cabeçalho TCP, e no nível da sessão, através do espaço de sequência de dados DSN incluído como opções do cabeçalho. Com esses dados é possível manter a consistência na transmissão em multi-fluxo, bem como o controle da janela de recepção.

Para ordenar os pacotes enviados e recebidos sobre múltiplos sub-fluxos em uma sessão, o protocolo MPTCP utiliza um número de sequência global de dados DSN (*Data Sequence Number*) de 64 bits. Com isso, é possível gerenciar a chegada de segmentos desordenados no nível da conexão, armazenando-os de forma consistente no nível do sub-fluxo, garantindo a entrega de dados ordenada para aplicação. Cada sub-fluxo, assim como em uma conexão TCP, tem seu próprio espaço de número de sequência de 32 bits, além de uma opção que mapeia seu espaço da sequência no espaço de sequência global da conexão MPTCP. Para tal, a opção `DATA_SEQUENCE_SIGNAL` no pacote TCP armazena o mapeamento de sequência de dados. Tal mapeamento é composto pelo número de sequência do sub-fluxo, número de sequência de dados e o comprimento dos dados. Um pacote `ACK` também pode ser enviado como opção, para confirmar o DSN recebido. Para tratamento de falhas nos sub-fluxos, pacotes perdidos podem ser retransmitidos por diferentes sub-fluxos, mantendo a consistência através do mapeamento DSN.

2.3.6 Controle de Congestionamento

O controle de congestionamento é um mecanismo originalmente utilizado pelo TCP para controlar a transmissão de dados do emissor ao longo da conexão. Os principais objetivos são: i) evitar perda de pacotes quando a rede está em estado de congestionando; ii) possibilitar uso justo de recursos quando uma conexão TCP compartilhar enlaces de gargalo no caminho fim-a-fim com outras transmissões. Tal controle é realizado através do ajuste dinâmico do tamanho da janela de congestionamento, w, durante a transmissão, determinando que menos ou mais pacotes sejam transmitidos em estado de congestionamento ou não na rede, respectivamente, conforme introduzido na Seção 2.2.

Cada sub-fluxo MPTCP, do ponto de vista de operação e gerenciamento de rede, é uma conexão TCP independente com sua janela própria de controle de congestionamento, medindo e estimando variáveis de estado, como RTT. Entretanto, simplesmente utilizar o controle de congestionamento padrão do TCP em cada um dos sub-fluxos MPTCP geraria um comportamento injusto quando esses sub-fluxos compartilhassem um mesmo enlace de gargalo com outros fluxos TCP de caminhos únicos (RAICIU; HANDLEY; WISCHIK, 2011). Portanto, cabe ao controle de congestionamento multi-fluxo regular, de forma acoplada, a expansão e retração da janela de congestionamento w_r de cada um dos sub-fluxos $r \in \mathcal{R}$.

Segundo Raiciu, Handley e Wischik (2011), as propriedades desejadas de um algoritmo de controle de congestionamento acoplado podem ser obtidas a partir de três objetivos:

1. **Melhorar a vazão.** Uma transmissão de multi-fluxos deve atuar, pelo menos, tão bem quanto uma transmissão de fluxo único em seu melhor caminho disponível.

2. **Não prejudicar.** Uma transmissão de multi-fluxos não deve tomar mais capacidade de quaisquer outros recursos compartilhados em um caminho que também esteja sendo usado por transmissões de fluxo único. Isto garante que a transmissão de multi-fluxos não irá prejudicar excessivamente outros fluxos.

3. **Balancear congestionamento.** Uma transmissão de multi-fluxos deve mover o máximo de tráfego possível dos caminhos mais congestionados para os menos congestionados. Esse objetivo permite que os dois primeiros possam ser obtidos.

Com alguma intersecção com os objetivos previstos em Raiciu, Handley e Wischik (2011), Peng et al. (2016) determinam três propriedades desejadas que um algoritmo de controle de congestionamento multi-fluxo deve buscar:

- **Responsividade**: quão rápido um algoritmo reage às mudanças que acontecem na rede, por exemplo, mudança do estado de um caminho congestionado para não congestionado.

- **Amistosidade**: um fluxo MPTCP é dito ser *amigável* ao TCP se o mesmo não dominar a banda disponível ao compartilhar a mesma rede com um fluxo TCP de caminho único.

- **Oscilação de janela**: que caracteriza a agressividade da expansão da janela de congestionamento em algoritmos de controle de congestionamento em estado de prevenção de congestionamento, por exemplo AIMD (*Additive Increase/Multiplicative Decrease*) utilizado no TCP.

Entretanto, diferente de Raiciu, Handley e Wischik (2011), os autores em Peng et al. (2016) utilizam essas propriedades para modelar um problema de otimização multi-objetivo. Não sendo possível otimizar as três propriedades ao mesmo tempo sem que uma ou outra saia

prejudicada, os autores apresentam modelo o qual possibilita encontrar um bom equilíbrio no comprometimento (*tradeoff*) entre as propriedades.

2.3.7 Escalonamento de Pacotes

Enquanto algoritmos de controle de congestionamento ajustam a operação de envio de todos os sub-fluxos $r \in \mathcal{R}$ em um controle acoplado para atingir objetivos diversos (por exemplo, justiça, responsividade, melhorar vazão, balancear congestionamento), um algoritmo de escalonamento no MPTCP é responsável por determinar, através de algum critério de decisão, qual sub-fluxo r será utilizado no envio de cada pacote. A política atualmente adotada pelo MPTCP realiza o escalonamento baseado nas estimativas correntes de tempo de ida e volta (`sRTT` - *smooth RTT*) dos sub-fluxos, considerando o sub-fluxo mais apto e rápido aquele que possuir a menor estimativa `sRTT` momento do envio do pacote. Maiores detalhes sobre algoritmos de escalonamento são apresentados no Capítulo 3.

Assim como os algoritmos de controle de congestionamento, o critério de escolha do sub-fluxo emissor é importante, pois é determinante no desempenho das transmissões em multi-fluxo. A decisão de escalonamento implica no uso mais ou menos frequente de sub-fluxos que permitirão o melhor ou o pior desempenho para a operação de envio, portanto, mais ou menos vazão de dados fim-a-fim.

No módulo de escalonamento, para cada sub-fluxo $r \in \mathcal{R}$, ocorrem as seguintes verificações preliminares:

1. **Inexistência**. Se o o sub-fluxo r não está nulo nas estruturas de dados do protocolo que definem uma conexão MPTCP, uma vez que variáveis internas da estrutura pode não ter sido inicializadas.

2. **Retransmissão de pacote perdido**. Se o pacote a ser transmitido está sendo re-injetado em r, ou seja, se o pacote será retransmitido sobre um mesmo sub-fluxo r que não conclui sua transmissão em um tentativa anterior.

3. **Estabelecimento incompleto**. Se o estabelecimento do sub-fluxo r está incompleto. Isso ocorre quando o estabelecimento de um novo sub-fluxo não é concluído, como por exemplo o pacote de controle é perdido ou recebido com erro. Na versão atual o MPTCP não tenta restabelecer o sub-fluxo (BONAVENTURE; PAASCH; DETAL, 2017).

4. **Indisponibilidade para transmissão**. Se o sub-fluxo r está em estado de perda ou com sua janela w_r cheia.

Em caso verdadeiro para qualquer uma dessas condições, o sub-fluxo r_i é considerado impróprio, sendo descartado para a transmissão do pacote, e parte-se para a verificação do próximo sub-fluxo $r_{i+1} \in \mathcal{R}$. Portanto, é aplicada uma dada política de escalonamento sobre

um sub-fluxo r se todas essas condições forem falsas para o mesmo, ou seja, somente se r for considerado apto para transmissão do pacote.

2.3.8 Gerenciamento de Caminhos

Entre cliente e servidor MPTCP, o gerenciador de caminhos é responsável pelas operações básicas de: estabelecimento do sub-fluxo mestre; sinalização dos endereços existentes nos dispositivos; adição e remoção de sub-fluxos existentes em uma conexão MPTCP; e encerramento da conexão MPTCP (FORD et al., 2011).

Cabe ao gerenciador determinar, através de alguma estratégia, a quantidade de sub-fluxos que serão estabelecidos sobre os caminhos fim-a-fim existentes entre os sistemas finais. Por exemplo, estabelecer um ou mais sub-fluxos por interface de rede (YEDUGUNDLA et al., 2016); estabelecer sub-fluxos mediante a troca de pacotes de controle com a opção `ADD_ADDR`, conforme novos caminhos forem sendo descobertos; estabelecer sub-fluxos apenas sobre os endereços anunciados no estabelecimento de sub-fluxo mestre.

2.3.9 Implementação do Protocolo

A principal implementação do MPTCP está disponível como módulo no núcleo do sistema operacional Linux e é mantida pelo projeto *MPTCP Linux Kernel Implementation*[1]. O protocolo pode ser instalado automaticamente através de gerenciadores de pacotes de *software*, com a ferramenta `apt-get` disponível em distribuições Linux baseadas em Debian. O código fonte do SO com o módulo do MPTCP está publicamente disponível para *download* através de repositório[2] do *github*. Essa iniciativa de código possibilita realizar estudos mais detalhados do protocolo, bem como propor e implementar melhorias no código e novas soluções, como escalonadores de sub-fluxos, controle de congestionamento e gerenciadores de caminhos.

A implementação modular do MPTCP consiste em um conjunto de extensões para o TCP (FORD et al., 2011), onde mecanismos para a execução de operações específicas do protocolo, conforme descrito nas seções anteriores, são implementados através de sub-módulos. A Figura 7 ilustra o conjunto dos sub-módulos no MPTCP.

Um dos principais sub-módulos do MPTCP é o de Controle que, conforme as definições de configurações realizadas, é responsável por inicializar os sub-módulos de gerenciamento de caminhos, escalonamento de pacotes e controle de congestionamento. No controle é possível realizar alterações importantes no protocolo, como adicionar novas variáveis de interesse dentro da estrutura de dados que representa o sub-fluxo, bem como inicializar essas variáveis quando o sub-fluxo é criado.

[1] <https://multipath-tcp.org>
[2] <https://github.com/multipath-tcp/mptcp>

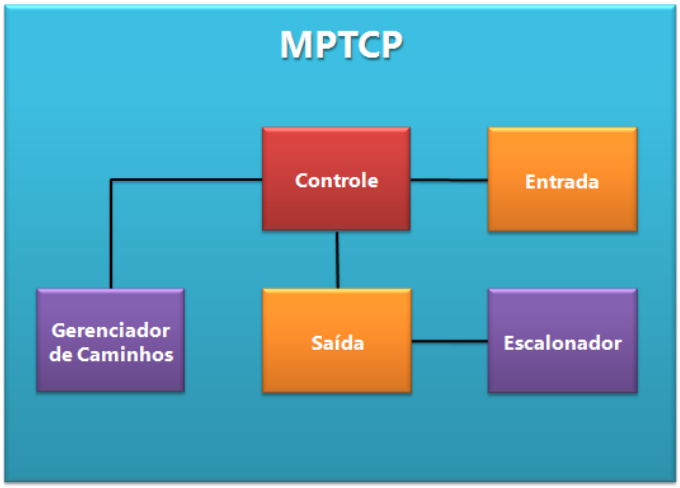


Figura 7 – Conjunto de sub-módulos do MPTCP.

O sub-módulo de gerenciamento de caminhos deve implementar a estratégia para descobrimento e estabelecimento de novos caminhos entre dois sistemas finais MPTCP. O sub-módulo de escalonamento, por sua vez, deve implementar a política de decisão de escolha do sub-fluxo disponível no momento da transmissão do pacote. Uma vez que estão implementados de forma modular, os gerenciadores e escalonadores podem ser substituídos em tempo de execução do sistema operacional.

Os sub-módulos de Entrada e Saída têm função semelhante aos do protocolo TCP. No receptor, o sub-módulo de entrada é responsável por gerenciar o recebimento de pacotes de dados enviados pela origem e pacotes de confirmação `ACK`s recebidos, mapeando a numeração de sequência dos pacotes em escopo global da conexão e local do sub-fluxo. No emissor, o sub-módulo de saída realiza a transmissão de novos pacotes e retransmissão de pacotes perdidos, sendo responsável por invocar o sub-módulo de escalonamento no momento do envio de cada pacote. É possível realizar alterações no código dos módulos de entrada e saída para, por exemplo, coletar dados de interesse sobre a conexão MPTCP, como o total de pacotes enviados e perdidos.

2.4 Considerações Finais

Neste capítulo, descrevemos na primeira parte os conceitos importantes para o contexto deste trabalho, como *multi-home*, *path*, *multi-Path*, sub-fluxo e conexão TCP. Apresentamos os mecanismos fundamentais do TCP, como o estabelecimento e encerramento de conexões, mecanismos que garantem a confiabilidade da transmissão TCP e o controle de congestionamento, que tem como função limitar a transmissão de bytes. Uma vez que o MPTCP consiste em um conjunto de extensões para o protocolo TCP, naturalmente várias operações se mantêm inalteradas no MPTCP, com exceção do controle de congestionamento. Apresentamos também os mecanismos internos do MPTCP e suas operações básicas, como o estabelecimento da conexão

de multi-fluxos, a operação de adição e remoção de sub-fluxos, o encerramento de conexão, o controle de fluxo, o controle de congestionamento, o escalonamento de pacotes e o gerenciamento de caminhos. Por fim, descrevemos os principais módulos que implementam esses mecanismos.

O conteúdo abordado neste capítulo então fornece ao leitor uma visão ampla da operação do MPTCP, partindo dos princípios de comunicação e mecanismos originados no TCP. Tal conteúdo é fundamental para o leitor compreender em qual contexto as soluções propostas neste trabalho se enquadram no funcionamento do protocolo MPTCP. Tendo em vista que o foco deste trabalho está no estudo do desempenho de transmissões em multi-fluxo MPTCP, maior atenção é destinada aos principais mecanismos que influenciam esse desempenho, como o controle de congestionamento, escalonamento de pacotes e gerenciamento de caminhos. Portanto, as soluções existentes em cada um desses mecanismos são tratadas como trabalhos relacionados. O capítulo seguinte apresenta em maior profundidade as principais soluções implementadas em cada um desses mecanismos.

3 TRABALHOS RELACIONADOS

Algoritmos de controle de congestionamento acoplado, políticas de escalonamento de pacotes e estratégias de gerenciamento de caminhos são os principais mecanismos que determinam a capacidade de transmissão em multi-fluxo MPTCP. Esses mecanismos possibilitam alcançar mais ou menos vazão na transmissão de dados, dependendo da forma como agregam as capacidades dos sub-fluxos em uma única conexão MPTCP. Assumimos esses mecanismos como trabalhos relacionados, uma vez que os mesmos interferem diretamente no desempenho das transmissões em multi-fluxos. Neste capítulo são apresentados detalhes do funcionamento de cada um deles.

3.1 Algoritmos de Controle de Congestionamento

A expansão e retração das janelas de congestionamento w_r de cada sub-fluxo que pertence a uma sessão $\mathcal{R}$ são dinamicamente ajustadas por um algoritmo que exerce controle sobre todos os sub-fluxos existentes. Atualmente, há quatro algoritmos de controle de congestionamento disponíveis no protocolo MPTCP:

- *Linked-Increases Algorithm* (LIA)
- *Opportunistic Linked-Increases Algorithm* (OLIA)
- *Balanced Linked Adaptation Algorithm* (BALIA)
- *Weighted Vegas* (wVegas).

As subseções a seguir detalham o funcionamento de cada um deles.

3.1.1 Linked-Increases Algorithm (LIA)

LIA (RAICIU; HANDLEY; WISCHIK, 2011) é o algoritmo de controle de congestionamento acoplado (*coupled*) utilizado como padrão no MPTCP, o qual foi adaptado para operar em multi-fluxos a partir do algoritmo *NewReno* (ALLMAN; PAXSON; BLANTON, 2009) de controle de congestionamento para fluxos de caminhos únicos no protocolo TCP. As adaptações em LIA foram realizadas apenas no estado de prevenção de congestionamento, mantendo inalteradas as operações previstas nas fases de partida lenta, retransmissão e recuperação rápida do TCP.

Considere $\mathcal{R}$ um conjunto de rotas definidas por caminhos fim-a-fim que compõem uma conexão MPTCP entre dois sistemas finais. Cada sub-fluxo $r \in \mathcal{R}$ possui uma variável de controle da janela de congestionamento, w_r. A capacidade total das janelas de congestionamento

dos sub-fluxos existentes é armazenada na variável $w_{\text{total}} = \sum_{r \in \mathcal{R}} w_r$. As variáveis $\ddot{\tau}_r$ e m_r armazenam o tempo de ida e volta (RTT - *round-trip time*) e o tamanho máximo do segmento (MSS - *Maximum Segment Size*) transmitido no sub-fluxo r, respectivamente.

A cada pacote `ACK` recebido no sub-fluxo r, a janela de congestionamento do sub-fluxo no lado emissor é atualizada da seguinte forma:

$$w_r = w_r + \min\left(\frac{\alpha \times b_r \times m_r}{w_{\text{total}}}, \frac{b_r \times m_r}{w_r}\right), \tag{3.1}$$

onde α é uma variável que armazena um fator atual de agressividade do sub-fluxo, e b_r armazena o número de bytes recentemente reconhecidos (`ACK`ed *bytes*) no sub-fluxo. Caso ainda não tenha ocorrido nenhum `ACK`, b_r receberá o valor de m_r.

Para cada perda de pacote no sub-fluxo r, a janela é diminuída por

$$w_r = w_r - \frac{w_r}{2}. \tag{3.2}$$

O cálculo de w_{total} não necessariamente é a soma dos valores w_r de cada sub-fluxo. Quando um fluxo está em estado de retransmissão rápida, o valor de w_r pode estar inflado e não representar o congestionamento real da janela. Nesse caso, utiliza-se o valor corrente do limiar da partida lenta `ssthresh` (*slow start threshold*), aqui representado por γ_r, dos fluxos em retransmissão rápida no cálculo de w_{total}.

Conforme Equação 3.1, a expansão w_r é então o mínimo entre: (primeiro argumento) um comprimento de janela acoplado para o todo fluxo MPTCP, considerando a agregação da capacidade dos sub-fluxos existentes $\forall r \in \mathcal{R}$; e (segundo argumento) um comprimento de janela calculado especificamente para o sub-fluxo r. O mínimo restringe cada janela a um aumento gradual, ponderando a agressividade conforme o fator atual em α.

O fator de agressividade é determinado por

$$\alpha = w_{\text{total}} \times \frac{\max\left(\frac{w_r}{\ddot{\tau}_r^2}\right)}{\left[\sum\left(\frac{w_r}{\ddot{\tau}_r}\right)\right]^2}, \tag{3.3}$$

observa-se que $\max(x_r)$ é o valor máximo para qualquer valor possível de r e $\sum(x_r)$ é o somatório para todos possíveis valores de r.

O fator α é, na prática, um agregador para o fluxo MPTCP que é baseado na vazão do sub-fluxo de melhor desempenho. Por exemplo, assumindo que os sub-fluxos em $\mathcal{R}$ tenham o mesmo $\ddot{\tau}_r$ (RTT) e m_r (MSS), a janela total w_{total} irá crescer por aproximadamente $\alpha \times m_r / \ddot{\tau}_r$.

Tal aumento é então distribuído para todos os sub-fluxos de acordo com o tamanho da janela instantânea w_r de cada um. Então, o sub-fluxo r irá expandir w_r por $(\alpha \times w_r / w_{\text{total}})/\ddot{\tau}_r$. Observa-se que, quando w_{total} é grande, o crescimento de w_r pode se aproximar a 0. Para contornar um possível efeito de inércia, o aumento mínimo é 1 nesse caso.

3.1.2 Opportunistic Linked-Increases Algorithm (OLIA)

O controle de congestionamento OLIA (KHALILI et al., 2013) foi proposto para resolver problemas de desempenho encontrados no LIA, contudo, sem perder seus benefícios. Assim como em LIA, OLIA implementa seu controle no estado de prevenção de congestionamento, mas também na atualização do limiar da partida lenta γ_r. Os outros mecanismos se mantém inalterados, como no TCP.

A operação em OLIA requer quantificar os bytes transmitidos com sucesso em cada sub-fluxo r. Uma aproximação do número de bytes transmitidos entre as duas últimas perdas é dado por

$$l_r = \max(l_{1r}, l_{2r}), \tag{3.4}$$

onde l_{1r} e l_{2r} contabilizam os bytes transmitidos com sucesso entre as últimas duas perdas e depois da última perda, respectivamente.

O algoritmo classifica os melhores sub-fluxos em um conjunto $\mathcal{B}$, os quais possuem os maiores valores de ${l_r}^2/\ddot{\tau}_r$, sendo $\ddot{\tau}_r$ o tempo de ida e volta (RTT) atualmente observado no sub-fluxo r. Há definição de outros dois conjuntos: $\mathcal{W}$ e $\mathcal{C}$. O primeiro contém os sub-fluxos com as maiores janelas w_r, enquanto o segundo coleta os melhores sub-fluxos em $\mathcal{B}$ que não estão em $\mathcal{W}$.

Considere $\mathcal{R}$ sendo o conjunto dos sub-fluxos existentes, a atualização do limiar da partida lenta `ssthresh` é dado por

$$\gamma_r = \begin{cases} 1, & \text{se } |\mathcal{R}| > 1, \\ 2, & \text{se } |\mathcal{R}| = 1, \end{cases} \tag{3.5}$$

sendo 1 MSS, caso houver mais de um sub-fluxo estabelecido, ou 2, como ocorre no TCP, se houver apenas um fluxo de caminho único. Isso evita que tráfego desnecessário seja transmitido sobre sub-fluxos congestionados quando existir mais de um sub-fluxo disponível.

Para cada `ACK` recebido no sub-fluxo r, a janela w_r é incrementada por

$$w_r = w_r + \left(\frac{w_r/\tau_r^2}{\left[\sum_{p \in \mathcal{P}} (w_p/\tau_p)\right]^2} + \frac{\alpha_r}{w_r} \right). \tag{3.6}$$

A percepção de congestionamento no sub-fluxo é percebida pela perda de pacotes, como no TCP. A cada perda de pacote no sub-fluxo r, a janela é então diminuída por dois, conforme Equação 3.2 em LIA.

A agressividade α é distinta para cada sub-fluxo r:

$$\alpha_r = \begin{cases} \frac{1/|\mathcal{R}|}{|\mathcal{C}|}, & \text{se } r \in \mathcal{C}, \\ \frac{1/|\mathcal{R}|}{|\mathcal{W}|}, & \text{se } r \in \mathcal{W} \text{ e } |\mathcal{C}| > 0, \\ 0, & \text{caso contrário.} \end{cases} \tag{3.7}$$

Assim, se todos sub-fluxos $r \in \mathcal{R}$ que estão no conjunto dos melhores sub-fluxos $\mathcal{B}$ possuírem as maiores janelas w_r, então $\alpha_r = 0$ para qualquer r, de modo que a capacidade agregada disponível é utilizada para todos os melhores sub-fluxos. Quando o conjunto $\mathcal{C}$ de sub-fluxos coletados não está vazio, existe pelo menos um sub-fluxo melhor em $\mathcal{B}$ com uma janela menor w_r. Então, α_r é maior para $\forall r \in \mathcal{C}$ e menor para $\forall r \in \mathcal{W}$. Isso faz com que w_r cresça mais rápido para $r \in \mathcal{B}$ e mais lentamente para $r \in \mathcal{W}$. Como resultado, o tráfego de sub-fluxos totalmente utilizados ($\mathcal{W}$) é redirecionado para sub-fluxos com capacidade livre disponível ($\mathcal{C}$).

3.1.3 Balanced Linked Adaptation Algorithm (BALIA)

A proposta do algoritmo BALIA (WALID et al., 2015) é trazer um bom balanceamento entre responsividade às mudanças da rede e amistosidade na concorrência dos enlaces com conexões de TCP, de fluxos únicos. De acordo com Walid et al. (2015), os algoritmos LIA e OLIA não possibilitam obter todos esses objetivos simultaneamente.

Assim como em OLIA, a operação em BALIA ocorre somente na fase de prevenção de congestionamento. Entretanto, o limiar mínimo de partida lenta γ_r é 1 MSS, ao invés de 2 MSS, quando houver mais de um sub-fluxo disponível.

Para cada `ACK` recebido no sub-fluxo r, a janela w_r é incrementada por

$$w_r = w_r + \left[\left(\frac{x_r}{\tau_r \times (\sum_{k \in \mathcal{R}} x_k)^2} \right) \times \left(\frac{1 + \alpha_r}{2} \right) \times \left(\frac{4 + \alpha_r}{5} \right) \right]. \tag{3.8}$$

Para cada pacote perdido, a janela é decrementada por

$$w_r = w_r - \frac{w_r}{2} \times \min \{\alpha_r, 1.5\}. \tag{3.9}$$

As variáveis x_r e α_r são definidas pelas Equações (3.10) e (3.11), respectivamente.

$$x_r = \frac{w_r}{\tau_r} \tag{3.10}$$

$$\alpha_r = \frac{\max(x_k)}{x_r} \tag{3.11}$$

O algoritmo de redução da janela em BALIA multiplica a redução prevista no TCP Reno, de $w_r/2$, por um fator no intervalo $[1, 1.5]$. Nesse caso, se houver apenas um sub-fluxo na conexão MPTCP, então $\alpha_r = 1$ e ambos incremento e decremento de BALIA serão reduzidos para as operações previstas no TCP Reno. Portanto, em transmissões de caminho único, quando $|\mathcal{R}| = 1$, BALIA irá operar como o TCP, regulando uma única janela de congestionamento.

3.1.4 Weighted Vegas (wVegas)

O controle de congestionamento wVegas (CAO; XU; FU, 2016) apresenta uma abordagem diferente dos algoritmos anteriores, LIA, OLIA e BALIA, que utilizam-se do evento de perda de pacote como forma de percepção de congestionamento. Em vez disso, wVegas se baseia em filas de atraso de pacote para determinar quão congestionado um sub-fluxo está. Dessa forma, garante-se uma sensibilidade maior às mudanças na rede, fazendo com que a janela w_r se ajuste mais rápidos (CAO; XU; FU, 2016).

O algoritmo wVegas também utiliza o parâmetro α_r como fator de agressividade, alterando a taxa de transmissão dos sub-fluxos para que o congestionamento no sub-fluxo individual r se estabilize. Ainda, o algoritmo utiliza a mesma estratégia do TCP de decremento multiplicativo de janela em caso de evento de perda. A principal operação do algoritmo ocorre na fase de prevenção de congestionamento. Na fase de partida lenta, a modificação é no limiar γ, que é ajustado para que a fase de prevenção de congestionamento possa ser ativada antes do esperado.

Para cada sub-fluxo r em transmissão, wVegas calcula a diferença δ_r, entre a taxa de envio esperada e a taxa de envio atual, da seguinte forma:

$$\delta_r = \left(\frac{w_r}{\hat{\tau}_r} - \frac{w_r}{\bar{\tau}_r} \right) \times \hat{\tau}_r, \tag{3.12}$$

onde $\bar{\tau}_r$ é a média do RTT no último ciclo de transmissões sobre o sub-fluxo r, e $\hat{\tau}_r$ é o RTT de base, sendo o menor RTT observado nas transmissões sobre r.

A condição para ingressar na fase de prevenção de congestionamento é decidida assim: caso o sub-fluxo r esteja em partida lenta e δ_r for maior que γ, então deve-se entrar na fase de prevenção de congestionamento; caso contrário, mantém-se a operação prevista no TCP.

Na fase de prevenção de congestionamento, se δ_r for maior ou igual a α_r, então ajusta-se o fator de agressividade do sub-fluxo r da seguinte forma:

$$\alpha_r = \omega_r \times \alpha_{\text{total}}, \tag{3.13}$$

onde ω_r é um peso e α_{total} é um parâmetro configurável que indica o total de pacotes a serem acumulados nas filas dos enlaces.

O peso é determinado por:

$$\omega_r = \frac{T_r}{\sum_{x \in \mathcal{R}}(T_x)}, \tag{3.14}$$

onde T é a taxa de transmissão atual do sub-fluxo que, considerando um sub-fluxo r, é determinada por

$$T_r = \frac{w_r}{\bar{\tau}_r}. \tag{3.15}$$

Durante a fase de prevenção de congestionamento a janela do sub-fluxo r é atualizada por incrementos ou decrementos de 1 MSS, conforme comparação de δ_r em relação a α_r:

$$w_r = \begin{cases} w_r - 1, & \text{if } \delta_r > \alpha_r, \\ w_r + 1, & \text{if } \delta_r < \alpha_r. \end{cases} \tag{3.16}$$

Por último, wVegas tenta drenar as filas dos enlaces com objetivo de melhorar também a precisão de $\hat{\tau}_r$. Para tanto, deve-se conhecer o atraso de fila q_r nos enlaces que um sub-fluxo r percorre, o qual é determinado pela diferença entre a média do RTT do ciclo de transmissão atual e o menor RTT observado, como por exemplo.

$$q_r = \bar{\tau}_r - \hat{\tau}_r. \tag{3.17}$$

O atraso de enfileiramento q_r somente é atualizado se a diferença atual $\bar{\tau}_r - \hat{\tau}_r$ for menor que o q_r conhecido. A janela de congestionamento w_r também é atualizada nesse momento, caso a diferença atual $\bar{\tau}_r - \hat{\tau}_r$ for duas vezes maior que o q_r conhecido. Nesse caso, a janela é atualizada da seguinte forma:

$$w_r = w_r \times \frac{\hat{\tau}_r}{2 \times \bar{\tau}_r}. \tag{3.18}$$

Em caso de perda de pacotes, a janela é atualizada normalmente como no TCP, conforme Equação (3.2) em LIA. Entretanto, wVegas atualiza a janela com maior sensibilidade às mudanças da rede e não somente pela percepção de congestionamento em evento de perda de pacote. Isso possibilita mais rápida convergência para um bom balanceamento de congestionamento entre os sub-fluxos, menor perda de pacotes e, consequentemente, menor ocupação dos *buffers* dos roteadores nos enlaces (CAO; XU; FU, 2016).

3.2 Políticas de Escalonamento de Pacotes

Havendo dados da aplicação disponíveis para transmissão no *buffer* de envio do MPTCP, o escalonador irá quebrar esse fluxo de *bytes* buferizado em pacotes e distribuir a carga de trabalho, selecionando o melhor sub-fluxo $r_{\text{best}} \in \mathcal{R}$ para enviá-lo (FORD et al., 2011). Essa seleção é determinada por alguma política de escalonamento, a qual faz com que pacotes sejam multiplexados sobre sub-fluxos de caminhos de características distintas, como por exemplo, diferentes latências, taxas de transmissão e taxas de perda.

Paasch et al. (2014b) descrevem dois problemas resultantes da multiplexação de pacotes sobre sub-fluxos heterogêneos. O primeiro é o bloqueio de *head-of-line*: como o MPTCP, sendo uma extensão do TCP, garante a entrega ordenada de dados para a aplicação, pacotes escalonados por sub-fluxos de baixa latência são buferizados no destino e têm que aguardar a chegada de pacotes que chegam fora de ordem por terem sido escalonados por sub-fluxos de alta latência. O segundo problema é o de *bufferbloat*: sob congestionamento, *buffers* excessivamente grandes em roteadores de enlaces de gargalo fazem com que pacotes fiquem enfileirados por longos períodos de tempo na rede. Ambos problemas causam impacto na latência de transmissão fim-a-fim.

Atualmente, há três escalonadores disponíveis no protocolo MPTCP:

- LL (*lowest Latency*), que é o escalonador padrão do MPTCP, cuja política é a de priorizar sub-fluxos de menor estimativa de RTT, τ_r (sRTT - *smoothed* RTT);
- RR (*Round-Robin*), que simplesmente seleciona um sub-fluxo depois do outro em uma ordem circular; e
- RDR (*Redundant Data Re-injection*), que replica o mesmo dado sobre todos os sub-fluxos disponíveis.

As próximas três sub-seções detalham o funcionamento de cada um deles. No final desta seção apresentamos outros escolanadores propostos na Literatura e discutimos as principais diferenças em relação aos escalonadores alternativos que propomos.

3.2.1 Lowest Latency (LL)

O escalonador de sub-fluxo padrão do MPTCP, LL, tem como política escolher o sub-fluxo de menor estimativa sRTT (*smoothed Round Trip Time*) de tempo de ida e volta. Essa política visa priorizar os sub-fluxos que estão possibilitando transmissões mais rápidas, para posteriormente escolher os sub-fluxos mais lentos para transmissão. Segundo Paasch et al. (2014b), decisões de escalonamento baseadas na menor latência podem mitigar os impactos dos problemas de bloqueio de *head-of-line* e *bufferbloat*.

O sRTT é uma variável bem conhecida no protocolo TCP e seu cálculo é determinado por

$$\text{sRTT}_{i+1} = (1 - \alpha) \times \text{sRTT}_i + \alpha \times \text{RTT}_i, \tag{3.19}$$

onde α é um peso, tipicamente de $1/8$ (JACOBSON, 1995), sRTT_i e RTT_i são a i-ésima estimativa e medição conhecidas do tempo de ida e volta. Observa-se, portanto, um peso maior para a estimativa i e um acréscimo menor para medição i do RTT. Entretanto, segundo o Algoritmo de Karn (KARN; PARTRIDGE, 1987), a amostragem de RTT não deve ser feita utilizando pacotes retransmitidos. Portanto, a estimativa de sRTT não é realizada sobre pacotes perdidos.

O escalonador LL é invocado toda vez que o emissor MPTCP for transmitir um pacote. Então, o escalonador seleciona o sub-fluxo de menor sRTT entre todos disponíveis na sessão do MPTCP, como descreve o Algoritmo 1. O sub-fluxo r_{best} será escolhido somente se estiver disponível, ou seja, enquanto sua janela w_r não estiver cheia e seu estado não estiver relacionado ao de perda de pacote. Conforme introduzido na Seção 1.1.1, identificamos em experimentos iniciais que o escalonador LL, ao sempre selecionar sub-fluxos de baixa latência através do menor sRTT, pode tomar decisões ruins de escalonamento que impactam o desempenho da transmissão multi-fluxo, principalmente quando os caminhos estão congestionados, sob frequente perda de pacotes e redução da janela de congestionamento.

Algoritmo 1: *Lowest Latency* (LL)

Input: Conjunto de sub-fluxos, $\mathcal{R}$
Output: Melhor sub-fluxo, $r_{\text{best}} \in \mathcal{R}$
$\tau_{\text{min}} \leftarrow$ `0xffffffff`
$r_{\text{best}} \leftarrow$ `NULL`
foreach $r \in \mathcal{R}$ **do**
  if $(\tau_r < \tau_{\text{min}})$ **then**
    $\tau_{\text{min}} \leftarrow \tau_r$
    $r_{\text{best}} \leftarrow r$
return r_{best}

3.2.2 Round Robin (RR)

RR é um escalonador que seleciona um sub-fluxo após o outro, conforme a política clássica de *round-robin*. Tal política pode garantir que a capacidade de cada sub-fluxo seja totalmente utilizada quando a distribuição de carga de trabalho sobre todos os sub-fluxos for igual (PAASCH et al., 2014b). A partir de um conjunto de sub-fluxos $\mathcal{R}$, a estratégia é percorrer pelos sub-fluxos e escolher um primeiro sub-fluxo r_{best} que esteja disponível, ou seja, que não esteja com sua janela cheia, não esteja em estado de perda e não já tenha sido escolhido no último ciclo. Conforme Algoritmo 2, a estratégia do escalonador é procurar um sub-fluxo através do *loop* que não tenha sido utilizado. Caso já tenha sido utilizado, este é marcado como *backup* reserva. Quando o sub-fluxo é selecionado este é adicionado no conjunto dos sub-fluxos já utilizados. O conjunto é esvaziado quando o algoritmo tiver percorrido todos os sub-fluxos existentes na conexão MPTCP.

Algoritmo 2: *Round-Robin* (RR)

```
Input: Conjunto de sub-fluxos, R
       Conjunto de sub-fluxos já utilizados, R*
Output: Melhor sub-fluxo, r_best ∈ R
r_best ← NULL
r_backup ← NULL
foreach r ∈ R do
    if (r ∈ R*) then
        r_backup ← r
        continue
    r_best ← r
if (r_best) then
    R* ← R* ∪ r_best
else if (r_backup) then
    R* ← ∅
    r_best ← r_backup
return r_best
```

Conforme discutido por Paasch et al. (2014b), em aplicações de transmissão massiva de dados (*bulk transfer*), o escalonamento em RR pode não se comportar exatamente em *round-robin*. Essas aplicações podem preencher as janelas de congestionamentos de todos o sub-fluxos, de modo que um novo pacote será escalonado somente quando houver espaço disponível em umas das janelas dos sub-fluxos. Isso gera um efeito conhecido como *ack-clock* (JACOBSON, 1995), ou seja, um novo pacote só será enviado após receber "clock" de ACKs que estão sendo esperados.

3.2.3 Redundant Data Re-injection (RDR)

Em contrapartida aos escalonadores existentes no MPTCP, o escalonador RDR não agrega a capacidade dos multi-fluxos (HUNGER; KLEIN, 2016), portanto, não aumenta o

desempenho com uma maior vazão. Esse escalonador tem como objetivo melhorar a confiabilidade na transmissão dos pacotes e equalizar o tempo de resposta na presença de falha nos enlaces de comunicação. Para isso, o escalonador envia o pacote primeiro pelo melhor sub-fluxo r_{best} utilizando política LL e, depois, replica o mesmo pacote sobre os outros sub-fluxos disponíveis na conexão MPTCP, conforme Algoritmo 3. Assim, o pacote que chegar mais rápido ao par de destino, possivelmente via r_{best} na ausência de falha, será o pacote contabilizado e todos os outros pacotes duplicados serão descartados como ocorre normalmente em conexões TCP.

Algoritmo 3: *Redundant Data Re-injection* (RDR)

Input: Conjunto pacotes a serem enviados, $\mathcal{P}$
Conjunto de sub-fluxos, $\mathcal{R}$

$r_{\text{best}} \leftarrow$ `NULL`
foreach $p \in \mathcal{P}$ **do**
  $r_{\text{best}} = \text{LL}(\mathcal{R})$
  send(p, r_{best})
  foreach $r \in \mathcal{R}$ **do**
    if $(r \neq r_{\text{best}})$ **then**
      send(p, r)

O escalonador RDR foi proposto para sistemas SCADA (*Supervisory Control and Data Acquisition*), os quais são utilizados para coletar dados em tempo real, controlar e monitorar serviços de equipamentos e infraestruturas vitais e críticas no domínio da indústria (LOPEZ et al., 2015). Ao ser projetado para garantir a confiabilidade das comunicações de infraestruturas críticas, RR não é apropriado para comunicação em redes de computadores em geral. Pacotes descartados no destino geram carga de trabalho inútil na rede, prejudicando o desempenho de outros fluxos nos enlaces de gargalo. Portanto, RDR piora a utilização de recursos de rede ao anular as principais propriedades de controle de congestionamento no MPTCP, de amistosidade e justiça no compartilhamento de enlaces de gargalos dos múltiplos caminhos fim-a-fim.

3.3 Gerenciadores de Caminhos

O gerenciador de caminhos (*path manager*) é responsável por detectar e sinalizar a existência de endereços IP entre os sistemas finais e, então, adicionar ou remover sub-fluxos sobre esses endereços em uma conexão MPTCP (FORD et al., 2011). Tal gerencimento implica em determinar quais e quantos sub-fluxos devem ser estabelecidos entre os sistemas finais ao longo da existência da conexão MPTCP. Há diferentes estratégias de gerenciamento para encontrar os possíveis caminhos entre os sistemas e estabelecer os sub-fluxos. Atualmente, os gerenciadores que estão implementados no MPTCP são (YEDUGUNDLA et al., 2016): *Default*, *Binder*, *nDiffPorts* e *Full-mesh*. Maiores detalhes de cada um são apresentados nas sub-seções a seguintes.

3.3.1 Default

Esse gerenciador, diferente do que seu nome sugere, não é o gerenciador configurado como padrão do protocolo. Conforme descrito nas configurações do MPTCP[1], esse gerenciador não faz nada, não descobre novos caminhos, tampouco os anuncia, e apenas aceita a criação de um novo sub-fluxo de forma passiva. Wang et al. (2017) observaram que o desempenho do gerenciador *default* não se difere muito do TCP, pois apenas um caminho é utilizado para transporte de dados, fazendo com que seu desempenho dependa da escolha apenas deste caminho caminho. O gerenciador *default* não aumenta a vazão de dados em comparação com o TCP. Entretanto, o gerenciador não impede que novos sub-fluxos sejam adicionados na conexão MPTCP, o que aumenta a resiliência com a redundância de outros sub-fluxos.

3.3.2 Binder

Este gerenciador estabelece um sub-fluxo sobre cada *gateway* existente em uma rede interna. Seu objetivo é direcionar o fluxo sobre esses caminhos, ao invés de direcionar para o destino final (BOCCASSI; FAYED; MARINA, 2013). Usando os caminhos ocultos por nós *relays*, permite-se que os dispositivos de usuários finais se beneficiem da agregação de banda sem que haja alteração na rede. Embora permita o uso de caminhos ocultos, o uso de *binder* é para cenários específicos, não sendo útil para aplicações em geral.

3.3.3 nDiffPort

O gerenciador *nDiffPort* cria n sub-fluxos sobre o mesmo par de endereços IP de origem e destino, modificando a porta de origem. Sua estratégia é similar ao protocolo *Stream Control Transport Protocol* (SCTP), ao promover múltiplos fluxos sobre um mesmo caminho. O número n pode ser manualmente configurado no sistema final[2]. A estratégia é estabelecer várias conexões sobre o mesmo caminho para sobrepor a limitação de banda imposta por *hardwares* que estão entre a origem e o destino. Ou seja, seu objetivo é imitar diferentes conexões TCP como se fosse de outra origem. Entretanto, o gerenciador nem sempre garante o aumento da vazão, uma vez que vários sub-fluxos de uma mesma conexão MPTCP irão concorrer pelos enlaces de gargalo em cada caminho.

3.3.4 Full-mesh

O *Full-mesh* é o gereciador padrão do MPTCP, o qual cria uma malha[3] de $n_s \times n_d$ sub-fluxos entre os sistemas finais, combinando os todos os n_s endereços IP de origem com

[1] <https://multipath-tcp.org/pmwiki.php/Users/ConfigureMPTCP>

[2] Há um arquivo no sistema final MPTCP, localizado em `/sys/module/mptcp_ndiffports/parameters/num_subflows`, onde é possível definir o número n de sub-fluxos desejados.

[3] A partir da versão `v0.90` do MPTCP é possível criar múltiplos sub-fluxo para cada par de endereços IPs definindo o número desejado em `/sys/module/mptcp_Full-mesh/parameters/num_subflow`.

todos os n_d endereços de destino (MEHANI et al., 2015). O gerenciador estabelece os sub-fluxos de forma estática, ou seja, se um sub-fluxo falhar durante a conexão MPTCP, este por sua vez, pode não ser restabelecido (BONAVENTURE; PAASCH; DETAL, 2017). O *Full-mesh* é abrangente, estabelecendo sub-fluxos sobre os diferentes caminhos existentes entre a origem e o destino. Isso permite maior desempenho, com aumento na vazão de dados quase sempre. O *Full-mesh* sobressai vantajosamente sobre todos os outros gerenciados de caminhos.

3.4 Considerações Finais

Neste capítulo, descrevemos detalhes das soluções existentes no protocolo MPTCP para os mecanismos de controle de congestionamento, escalonamento de pacotes e gerenciamento de caminhos. O controle de congestionamento exerce enorme influência na transmissão em multi-fluxos, pois limita a transmissão de bytes sobre cada sub-fluxo através do dimensionamento da janela de congestionamento, diminuindo ou aumentando com base no congestionamento identificado. O escalonamento de pacotes, nesse contexto, é fundamental, pois ao designar pacotes os sub-fluxos existentes, determina a carga de trabalho de cada sub-fluxo. O escalonador de pacotes escolhe o sub-fluxo com base em algum critério de decisão, a escolha do sub-fluxo influencia no desempenho pois os sub-fluxos podem ter distinção de características. Se o escalonador der prioridade para um sub-fluxo de menor capacidade de transmissão ao invés de um de maior, a vazão total da conexão MPTCP pode estar distante da soma das capacidades dos caminhos existentes. O gerenciador de caminhos, igualmente importante, é responsável por identificar os caminhos disponíveis entre os sistemas finais e estabelecer sub-fluxos sobre eles. Esse gerenciador também determina a quantidade de sub-fluxos que serão estabelecidos sobre cada caminho. Um sub-fluxo estabelecido sobre um mesmo caminho já utilizado por outro sub-fluxo causará concorrência pelo seu uso do respetivo enlace de gargalo, podendo degradar ainda mais o desempenho da conexão MPTCP.

O conteúdo deste capítulo fornece ao leitor uma descrição detalhada das soluções existentes que influenciam diretamente o desempenho das transmissões em multi-fluxos. No contexto das soluções propostas neste trabalho, compreender o funcionamento das soluções atualmente implementadas nos mecanismos internos do MPTCP foi fundamental para entender como e porquê os problemas identificados ocorreram. Ao compreender o funcionamento de cada um dos algoritmos, foi então possível propor novas soluções, sem que fosse necessário a implementação de algoritmos complexos. As soluções propostas são apresentadas com maior profundidade no capítulo seguinte.

4 SOLUÇÕES PROPOSTAS

Face aos problemas de desempenho que identificamos nas soluções padrão do MPTCP, conforme introduzido na Seção 1.1, investigamos soluções alternativas de escalonamento de pacote e gerenciamento de caminhos. Neste capítulo, são descritas em maiores detalhes as quatro soluções propostas para aprimorar o protocolo MPTCP e melhorar o desempenho de transmissões multi-fluxos:

- três algoritmos alternativos de escalonamento - HSR (*Highest Sending Rate*), LWS (*Lagest Window Space*) e LTS (*Lowest Time/Space*);
- e um novo gerenciador de caminhos, *Single-mesh* (SM).

As seções seguintes discutem os algoritmos e as implementações de cada solução. As considerações práticas, de implementação de cada solução proposta, podem ser encontradas no Apêndice A.

4.1 Políticas Alternativas de Escalonamento

Como introduzido na Seção 1.1, o escalonador LL, padrão do MPTCP, pode vir a apresentar seleção inconsistente de sub-fluxos em alguns cenários de comunicação, de taxas de perda de pacotes mais elevados em caminhos congestionados. Decisões ruins de escalonamento geram impacto negativo no desempenho de um fluxo MPTCP, resultando em uma menor vazão que a capacidade agregada dos múltiplos fluxos poderia atingir sob decisões acertadas.

4.1.1 Largest Window Space (LWS)

LWS é uma solução proposta de escalonamento de sub-fluxos que utiliza como critério de escolha o valor corrente de espaço na janela de congestionamento, ws (*window space*). Tal espaço é determinado pela diferença entre o tamanho da janela de congestionamento em w_r e a quantidade de pacotes enviados mas pendentes de confirmação do destinatário (*in-flight packets*), sendo utilizado frequentemente pelo escalonadores para verificar a disponibilidade do sub-fluxo em relação ao tamanho do pacote a ser enviado. A hipótese considerada para a proposição de LWS foi a seguinte:

Hipótese 1 (*Window Space*) *Sub-fluxos que percorrem enlaces de maior capacidade tendem a ter maior espaço em suas janelas de congestionamento, pois o envio de pacotes da janela é mais rápido, logo, tendem a ter uma menor quantidade de pacotes pendentes de confirmação (in-flight), f_r, na janela w_r.*

Partindo da Hipótese 1, o escalonador LWS procura pelo melhor sub-fluxo r_{best}, aquele com maior ws entre todos os sub-fluxos r estabelecidos em uma sessão $\mathcal{R}$ do protocolo MPTCP, conforme descreve o Algoritmo 4.

Algoritmo 4: *Largest Window Space* (LWS)

Input: Conjunto de sub-fluxos, $\mathcal{R}$
Output: Melhor sub-fluxo, $r_{\text{best}} \in \mathcal{R}$
$\tau_{\text{min}} \leftarrow$ `0xffffffff`
$ws_{\text{max}} \leftarrow$ `0`
$r_{\text{best}} \leftarrow$ `NULL`
foreach $r \in \mathcal{R}$ **do**
  $ws \leftarrow w_r - f_r$
  if $(ws \leq 0$ **and** $\tau_r \leq \tau_{min})$ **then**
    $\tau_{\text{min}} \leftarrow \tau_r$
    $r_{\text{best}} \leftarrow r$
  else if $(ws \geq ws_{\text{max}})$ **then**
    $ws_{\text{max}} \leftarrow ws$
    $r_{\text{best}} \leftarrow r$
return r_{best}

Importante observar que o cálculo do espaço da janela ws pode resultar em números negativos. Conforme algoritmos de controle de congestionamento discutidos no Capítulo 3, a estratégia de diminuição multiplicativa baseada em fator de 2 do TCP é preservada nos algoritmos do MPTCP. Em estado de perda de pacotes, a diminuição pode resultar em um tamanho de janela menor ou igual à quantidade de pacotes *in-flight*, portanto, resultando em um valor negativo ou nulo de espaço de janela ws. Nessa situação, o escalonador LWS reduz sua operação, operando com o escalonamento em LL. Isso torna o LWS um algoritmo híbrido, alternando a decisão de escalonamento por diferentes critérios.

Durante o laço de escalonamento, pacotes *in-flight* pendentes de confirmação podem mudar o valor das variáveis de estado observadas nos sub-fluxos. Nesse caso, ao verificar a condição de $\leq$ e $\geq$ nas comparações, prioriza-se o sub-fluxo mais recente, dentre os que possuem a menor latência em τ_{min} ou maior espaço em ws_{max}

4.1.2 Lowest Time/Space (LTS)

LTS é uma solução proposta para escalonamento de sub-fluxos que define, como critério de escolha, a menor relação entre o valor da estimativa do tempo de ida e volta, τ_r (latência em sRTT), e o valor corrente de espaço na janela de congestionamento, ws (*window space*). A hipótese considerada na proposição de LTS foi a seguinte:

Hipótese 2 (*Time/Space*) *Há sub-fluxos que utilizam enlaces de maior vazão e tendem a ter maior espaço em suas janelas de congestionamento devido à rápida transmissão dos pacotes.*

Há sub-fluxos que percorrem caminhos menos congestionados e, portanto, tendem a ter uma menor latência de transmissão. Nesse caso, um maior desempenho pode ser obtido ao priorizar sub-fluxos de menor latência e, ao mesmo tempo, de maior espaço na janela, ou seja, sub-fluxos que possuem menor relação entre essas variáveis.

Partindo da Hipótese 2, o escalonador LTS propõe a escolha do melhor sub-fluxo r_{best}, aquele com menor τs entre todos os sub-fluxos r estabelecidos em uma conexão $\mathcal{R}$ do protocolo MPTCP, conforme descreve o Algoritmo 5.

Algoritmo 5: *Lowest Time/Space* (LTS)

```
Input: Conjunto de sub-fluxos, R
Output: Melhor sub-fluxo, r_best ∈ R
τ_min ← 0xffffffff
τs_min ← 0xffffffff
r_best ← NULL
foreach r ∈ R do
    ws ← w_r − f_r
    if (ws ≤ 0 and τ_r ≤ τ_min) then
        τ_min ← τ_r
        r_best ← r
    else
        τs ← τ_r / ws
        if (τs ≤ τs_min) then
            τs_min ← τs
            r_best ← r
return r_best
```

Assim como LWS, faz-se primeiro o cálculo e a verificação do espaço da janela, evitando inconsistência no seu uso a partir de resultados negativos. Nesse caso, se o valor de espaço em ws for menor que zero, deve ser utilizado a comparação e escolha do sub-fluxo r com menor τ_r, portanto, reduzindo a operação para LL, assim como em LWS.

4.1.3 Highest Sending Rate (HSR)

A solução proposta em HSR é um escalonamento que utiliza como critério de escolha a atual capacidade de transmissão dos sub-fluxos. Para a proposição de HSR, a hipótese considerada foi a seguinte:

Hipótese 3 (*Sending Rate*) *Sub-fluxos que possuem maior taxa atual de transmissão podem ser os mais aptos para o envio de pacotes, uma vez que a relação de maior janela w_r e menor RTT instantâneo em $\ddot{\tau}$ indica um caminho fim-a-fim não congestionado de maior capacidade.*

Partindo da Hipótese 3, o escalonador HSR calcula a taxa atual de envio, sr (*sending rate*), através da razão entre o tamanho da janela de congestionamento w_r em bytes e a medição

instantânea $\ddot{\tau}$ do tempo de ida e volta (RTT). O valor de janela atual do sub-fluxo r em *bytes* é obtido pela multiplicação por m_r, que indica o tamanho máximo do segmento (MSS) que está sendo utilizado no sub-fluxo. O escalonador então procura pelo sub-fluxo de maior capacidade de transmissão entre os sub-fluxos existentes $\mathcal{R}$ na conexão MPTCP, conforme descreve o Algoritmo 6.

Algoritmo 6: *Highest Sending Rate* (HSR)

Input: Conjunto de sub-fluxos, $\mathcal{R}$
Output: Melhor sub-fluxo, $r_{\text{best}} \in \mathcal{R}$
$sr_{\text{max}} \leftarrow 0$
$r_{\text{best}} \leftarrow \texttt{NULL}$
foreach $r \in \mathcal{R}$ **do**
    $sr \leftarrow w_r \times m_r / \ddot{\tau}_r$
    if $(sr \geq sr_{\text{max}})$ **then**
        $sr_{\text{max}} \leftarrow sr$
        $r_{\text{best}} \leftarrow r$
return r_{best}

A capacidade de transmissão, sr, uma medida instantânea que determina a taxa atual que o sub-fluxo consegue enviar no momento que precede o envio do pacote. Assim como LWS e LTS, o valor das variáveis de estado observadas nos sub-fluxos pode ser alterado pela confirmação ou não de pacotes *in-flight*. Ao verificar a condição de igualdade ($\geq$) nas comparações, HSR prioriza o sub-fluxo mais recente dentre os que possuem a maior taxa sr_{max}.

A seção a seguir apresenta solução proposta, *Single-mesh*, para no escopo do gerenciamento de caminhos.

4.2 Novo Gerenciador de Caminhos

O gerenciador de caminhos tem como objetivo identificar e sinalizar a presença de caminhos fim-a-fim e, conforme à estratégia selecionada, estabelecer sub-fluxos MPTCP sobre eles (FORD et al., 2011). O gerenciador padrão do MPTCP é o *Full-mesh*, conforme descrito na Seção 3.3, o qual prevê o estabelecimento de uma malha completa de sub-fluxos de acordo entre todos n_s e n_d endereços existentes nos sistemas finais de origem e destino, respectivamente.

Ao estabelecer $\max(n_s, n_d)$ sub-fluxos sobre um mesmo caminho fim-a-fim, a estratégia em *Full-mesh* gera concorrência entre os próprios sub-fluxos no uso compartilhado do enlace de gargalo desse caminho. Como introduzido na Seção 1.1.2, em caminhos confiáveis e não congestionados, observamos que o impacto da disputa dos sub-fluxos pelo maior uso do enlace ocorre na forma de degradação de desempenho de toda conexão MPTCP, resultando em menor vazão que uma conexão TCP de caminho único.

4.2.1 Single-mesh (SM)

O SM é um gerenciador de malha simples que tem o objetivo de evitar o congestionamento entre sub-fluxos que compartilham um mesmo caminho fim-a-fim. Dado o problema identificado na concorrência inter-sub-fluxos, a hipótese de SM é dada por:

Hipótese 4 (Single-mesh) *Uma malha simples limitada a um sub-fluxo por caminho fim-a-fim evita que sub-fluxos r de uma mesma conexão $\mathcal{R}$ disputem pelo uso de um mesmo enlace de gargalo, podendo, com isso, melhorar o desempenho agregado de todo multi-fluxo em cenários de baixo congestionamento.*

A operação do gerenciador SM ocorre no lado cliente da comunicação, uma vez que é a entidade ativa que inicia o estabelecimento dos sub-fluxos nas conexões MPTCP. A malha simples é então definida pelo estabelecimento de apenas um sub-fluxo por rede IP do cliente, conforme descreve no Algoritmo 7. No estabelecimento de sub-fluxos com os endereços $d \in \mathcal{D}$ do destino, para cada endereço IP de origem s das redes $\mathcal{S}$ nas quais as interfaces do cliente estão endereçadas, verifica se s já não está sendo utilizado por um dos sub-fluxos existentes em $\mathcal{R}$. Caso o endereço s ainda não esteja sendo utilizado, cria-se então um novo sub-fluxo r sobre o caminho (s, d).

Algoritmo 7: *Single-mesh* (SM)

```
Input: Conjunto de endereços de origem (cliente), S
       Conjunto de endereços de destino (servidor), D
Output: Conjunto de sub-fluxos estabelecidos pelo Full-mesh,
        R
foreach d ∈ D do
    r_exist ← false
    foreach s ∈ S do
        S_R ← get_source_addresses(R)
        if (s ∈ S_R) then
            r_exist ← true
            break loop
    if (not r_exist) then
        r ← create_subflow(s, d)
        R ← R ∪ r
return R
```

4.3 Comparação Qualitativa com Soluções Propostas na Literatura

De acordo com Arzani et al. (2014), pode-se melhorar a vazão fim-a-fim aumentando o tamanho do *buffer* de envio. Em seus experimentos, os autores observaram que, além de um *buffer* de envio maior trazer melhorias significativas na vazão da aplicação, a escolha dos

caminhos através de suas características é fundamental para seleção do melhor caminho. Assim como discutido por Arzani et al. (2014), nossas soluções alternativas utilizam características de cada caminho para escolha do melhor caminho.

Paasch et al. (2014b) apresentam duas extensões do escalonador LL para diminuir os impactos dos problemas de *head-of-line* e *bufferbloat*, introduzidos na Seção 3.2. O primeiro escalonador estendido de LL, chamado RP (*Retransmition and Penalization*), realiza retransmissão oportunística, re-injetando o pacote que causou bloqueio de *head-of-line* no sub-fluxo que tem espaço disponível em janela de congestionamento. Isso possibilita que situações de bloqueio de *head-of-line* sejam superadas rapidamente, além de compensar as diferenças de RTT entre os sub-fluxos. A penalização em RP ocorre pela redução da janela de congestionamento do sub-fluxo de maior latência, consequentemente, reduzindo a taxa de envio e o efeito de *bufferbloat* desse sub-fluxo. O segundo escalonador estendido de LL, chamado BM (*Bufferbloat Mitigation*), identifica sub-fluxos em situações de *bufferbloat* e reduz a quantidade de dados a serem enviados sobre esses sub-fluxos, impondo uma janela de congestionamento de limite para os mesmos. Para um sub-fluxo r, *bufferbloat* é identificado pela maior diferença entre o mínimo sRTT estimado ($\tau_{r_{\text{min}}}$) e o sRTT corrente (τ_r). O limite da janela de congestionamento do sub-fluxo é então determinado por $w_{r_{\text{limit}}} = \lambda \times (\tau_{r_{\text{min}}}/\tau_r) \times w_r$, onde λ determina a tolerância entre o sRTT mínimo e o sRTT corrente. Diferente das soluções alternativas que propomos para o protocolo MPTCP, ambos escalonadores, RP e BM, não foram projetados com o objetivo de melhorar a vazão multi-fluxo, mas sim de reduzir o atraso e os requisitos de tamanho de *buffer* de recepção.

O desempenho da transmissão em multi-fluxo é resultante da cooperação entre escalonamento e controle de congestionamento. Yang, Amer e Ekiz (2013) observaram que o escalonador LL, padrão do MPTCP, falha com o uso das amostras de RTT para a retransmissão de pacotes perdidos, não permitindo determinar precisamente a capacidade dos caminhos. Outro problema observado é o incremento excessivo da janela de congestionamento no algoritmo de LIA, que induz maior perda de pacotes, uma vez que pode ser transmitido no sub-fluxo mais pacotes do que suporte a sua a capacidade real. Os autores então propõem um escalonador que utiliza estimativa da capacidade do caminho, levando em consideração uma capacidade máxima. Os resultados obtidos demonstram ganho de vazão e diminuição de pacotes retransmitidos. Nosso estudo envolve a utilização de outras características dos sub-fluxos para tomar a decisão de escalonamento, por exemplo, estimando a capacidade atual do caminho através da relação entre a janela de congestionamento corrente e o RTT instantâneo.

Outro trabalho envolvendo escalonadores é o de Singh et al. (2012). Suas propostas apresentam vários escalonadores utilizando diversas variáveis estatísticas encontradas no sub-fluxo, tais como: tempo de chegada; capacidade do sub-fluxo; atraso de propagação seguinte no sub-fluxo; vazão; tamanho da fila no sub-fluxo; tempo de espera na fila do sub-fluxo; atraso na transmissão do sub-fluxo; atraso de envio do sub-fluxo; atraso de reordenamento; número

de pacotes consecutivos escalonados no sub-fluxo. Os autores apontam o escalonador *Minimum Delivery Delay* como melhor escalonador nos seus resultados. Entretanto, os autores não comparam suas propostas com os escalonadores implementados no MPTCP, não sendo possível realizar comparações mais amplas no Estado da Arte, com outros escalonadores propostos por outros autores. Nosso trabalho envolve um ambiente experimental que possibilita caracterizar condições variadas e mais complexas de transmissão fim-a-fim, onde as soluções que propomos de escalonamento alternativo foram avaliadas e comparadas com as solução padrão do protocolo.

Hwang e Yoo (2015) descrevem o problema de queda de desempenho no envio de mensagens curtas utilizando o MPTCP em comparação ao TCP comum. O problema ocorre quando o escalonador LL escolhe apenas os sub-fluxos disponíveis. Logo, se houver um sub-fluxo rápido indisponível e um sub-fluxo lento disponível, o escalonador irá escolher o sub-fluxo lento. Os autores observaram que essa estratégia oferece desempenho ruim para transmissão de mensagens curtas. Para solucionar este problema, os autores propõem o escalonador *freeze*, que congela temporariamente o envio de pacotes sobre os sub-fluxos mais lentos e, assim, priorizando o envio através dos sub-fluxos mais rápidos, de menor latência. O escalonador *freeze* obteve menor tempo de transmissão de mensagem em relação ao TCP comum. Nosso estudo não visa melhorar o desempenho do envio de mensagens curtas no MPTCP, mas de transmissão massiva de dados, de *bulk transfer*, onde pode-se obter maior vazão com decisões alternativas de escalonamento, em comparação ao LL.

O MPTCP pode sofrer com a chegada de pacotes fora de ordem no nó de destino, uma vez que sub-fluxos podem ter latências diferentes. Yang, Wang e Amer (2014) observaram que esse problema impacta negativamente o desempenho de aplicações sensíveis à latência. Para solucionar, é proposto um escalonador que envia menos pacotes num sub-fluxo com maior latência e mais pacotes em sub-fluxo com menor latência. Os pacotes são escalonados mesmo que não haja espaço na janela de congestionamento. Se os pacotes não forem descartados pela rede, todos os pacotes devem chegar ordenados no destino. A solução dos autores estima a quantidade de pacotes que devem ser enviados por cada sub-fluxo, determinando a capacidade de transmissão que um sub-fluxo permite, para que se evite descarte de pacotes na rede. Os resultados obtidos pela proposta de Yang, Wang e Amer (2014) foram melhores que o escalonador LL em experimentos com aplicações à latência, possibilitando também maior vazão para essas aplicações. Nossas propostas de escalonamento apresentaram melhor desempenho que LL, possibilitando maior vazão para aplicações de *bulk-transfer* na maior parte dos experimentos realizados a partir de cenários variados de comunicação fim-a-fim. Dessa forma, nosso foco de estudo não foi melhorar a transmissão de aplicações sensíveis a latência.

O *escalonador* proposto por Ke et al. (2016) foi implementado para solucionar o problema de bloqueio de *buffer*, este problema ocorre pelo limitação do *buffer* de recebimento, e especialmente quando os caminhos tem características muito diferentes, como por exemplo

latência. Esse problema ocorre devido ao envio excessivo de pacotes aos sub-fluxos. O escalonador MPTCP-MA (*Multiple Attribute-aware*) foi implementado utilizando três módulos. O primeiro módulo monitora as informações de estados dos sub-fluxos, tais como o RTT e a janela de congestionamento, para refletir a qualidade de cada sub-fluxo. O segundo módulo ordena em uma lista todos os sub-fluxos com base no menor RTT e, caso ocorra uma comparação igual, é usado a janela de congestionamento como comparação, sendo escolhido o sub-fluxo com maior janela. O último módulo é o escalonador em si, que é responsável por enviar os pacotes utilizando a lista de sub-fluxos ordenada. Nos resultados, a vazão obtida com o escalonador MPTCP-MA obteve um bom resultado, pois além de reduzir a chegada de pacotes desordenados no destino, sua vazão foi um pouco maior que os escalonadores comparados. Nos escalonadores propostos em nosso trabalho, utilizamos as variáveis de estado dos sub-fluxos, sem realizar modificações significativas na estrutura de dados dos escalonadores do MPTCP. Ainda avaliamos os escalonadores sobre condições de rede mais diversificadas.

O problema de recebimento de pacotes desordenados é agravado com a transmissão sobre caminhos de características heterogêneas no MPTCP, impactando no tempo de transmissão fim-a-fim. Ni et al. (2014) apresentam um escalonador que trata esse problema levando em conta a perda de pacotes. Para a coleta desse dado, é observado o estado dos sub-fluxos a cada ocorrência do evento de perda de pacote, desconsiderando os pacotes que foram retransmitidos para que o cálculo da perda seja estimado com mais precisão. A perda é estimada adotando a estratégia de TCP Reno. Com base na janela de congestionamento, RTT e na perda de pacote, o escalonador estima quanto de dado pode ser transmitido com sucesso sobre cada sub-fluxo. O escalonador proposto em Ni et al. (2014) mostrou ser mais eficiente na transmissão, melhorando a vazão global e solucionando o problema de recebimento de pacotes desordenados. Nossos escalonadores não estimam a perda de pacotes nos sub-fluxos, o que pode ser investigado em um trabalho futuro.

Oh e Lee (2015) descrevem a situação de degradação de vazão quando os sub-fluxos apresentam diferentes capacidade de transmissão e latência, causando longos intervalos entre os pacotes enviados sobre sub-fluxos heterogêneos. Ou seja, pacotes transmitidos por sub-fluxos de maior latência podem ser mais lentos, que esperar um sub-fluxo de menor latência estar disponível para envio. Para solução desse problema, é proposto o escalonador *Constraint-based Proactive* (CP) que leva em consideração o *buffer* do receptor e a latência dos sub-fluxos. Pacotes são escalonados a partir da comparação do desempenho dos sub-fluxos, o que é determinado pelo número estimado de pacotes fora de ordem e a capacidade de recepção do *buffer* no destinatário. Foram propostas duas estratégias em CP: a *radical* e a *conservative*. Na primeira estratégia, os pacotes são designados para o *buffer* de envio do sub-fluxo mais rápido até que sua janela esteja cheia. Os pacotes são designados para o próximo sub-fluxo somente quando o *buffer* disponível for maior que o sub-fluxo mais rápido. A estratégia *conservative* ocorre de forma semelhante ao *radical* na primeira etapa, onde os pacotes são designados ao *buffer* de envio do sub-fluxo mais rápido até que a janela esteja cheia. Se o sub-fluxo mais rápido estiver

no estado de partida lenta ou o número de pacotes *outstanding* for maior que metade da janela do receptor, então o pacote é transmitido pelo sub-fluxo mais rápido. Caso contrário, os pacotes são designados para o próximo sub-fluxo que possuir *buffer* disponível. O escalonador CP solucionou o problema quando houve diferença das características entre os sub-fluxos, utilizando os sub-fluxos com alta heterogeneidade mais adequadamente, permitindo aumentar a vazão fim-a-fim (OH; LEE, 2015). Nosso trabalho também propõe soluções de escalonamento que possibilitam ganho de desempenho em cenários de transmissão sobre sub-fluxos de características heterogêneas. Entretanto, o CP aborda duas estratégias, comparando o *buffer* disponível e a capacidade estimada para determinar quando um pacote deve ser designado para um sub-fluxo. Nossas propostas são oriundas de algoritmos muito simples, que apenas escolhem o sub-fluxo com base na comparação das características dos sub-fluxos.

Le e Bui (2015) também investigam os impactos da entrega de pacotes fora de ordem devido aos sub-fluxos possuírem características heterogêneas. Os autores propõem o escalonador *forward-delay-based packet* (FDPS), onde é estimado o atraso do pacote encaminhado em cada sub-fluxo, selecionando o sub-fluxo de menor estimativa. A estimativa é realiza através do envio de dois pacotes, cada um sobre sub-fluxos diferentes. Uma vez conhecendo o atraso do pacote encaminhado de cada sub-fluxo, calcula-se a diferença entre eles como forma de comparação. Essa operação é realizada para todos sub-fluxos, sendo escolhido o sub-fluxo com menor diferença. O escalonador FDPS reduz significativamente a chegada de pacotes desordenados no *buffer* do receptor. Nosso trabalho não tem foco na solução do problema de pacotes fora de ordem no receptor.

De acordo com Tsai, Chou e Lan (2016), o escalonador LL sofre do problema de saturação janela do sub-fluxo de menor latência, mesmo havendo espaço disponível para transmissão nas janelas de congestionamento dos outros sub-fluxos na conexão MPTCP. Esse problema gera maior atraso na transmissão dos pacotes, pois tende a utilizar sempre um dado melhor sub-fluxo e este pode gerar atraso no envio dos pacotes por ter sua janela sempre cheia. A ideia para diminuir este problema é distribuir a carga entre os sub-fluxos. O escalonador proposto pelos autores estima o atraso de cada sub-fluxo através da diferença entre a soma do *timestamp*, tempo de espera e sRTT entre um pacote e outro. O escalonador resolve o problema de saturação de sub-fluxo. Contudo, não se evita a reordenação dos pacotes, gerando degradação da vazão do MPTCP. Nossos escalonadores foram propostos para aumentar a vazão do MPTCP, ou seja, não têm o objetivo de tratar o problema de saturação de sub-fluxo.

4.4 Considerações Finais

Neste capítulo, descrevemos as soluções propostas para os problemas de desempenho inicialmente identificados. No escalonamento, definimos hipóteses para cada escalonador alternativo. O escalonador LWS foi proposto usando o critério do sub-fluxo de maior espaço na

janela de congestionamento, nos permitindo explorar caminhos com maiores janelas de congestionamento, e ao mesmo tempo pacotes que ainda não foram confirmados por ACK. O escalonador LTS foi proposto usando o critério de decisão baseado no sub-fluxo de menor razão entre o sRTT atual e o espaço na janela de congestionamento. Tal relação é importante para identificar e priorizar caminhos com menor latência e também com maior espaço na janela de congestionamento. O escalonador HSR, com objetivo de priorizar caminhos com alto *goodput* atual, foi proposto usando como critério de decisão o sub-fluxo de maior razão entre a janela de congestionamento atual e o RTT instantâneo. No gerenciador de caminhos, também definimos a hipótese para a solução proposta. O Single-mesh foi proposto com objetivo de garantir que apenas um único sub-fluxo seja estabelecido sobre o mesmo caminho, com isso diminuímos a concorrência no uso de caminhos disponíveis.

Nesse contexto, o conteúdo abordado neste capítulo provê ao leitor uma visão ampla das soluções propostas e qual objetivo de cada uma delas. Para o contexto dos experimentos, permite ao leitor compreender os resultados esperados de cada experimento realizado das soluções propostas. O capítulo seguinte aborda os experimentos realizados com as soluções propostas.

5 AVALIAÇÃO EXPERIMENTAL

Neste capítulo apresentamos a análise experimental realizada para avaliar o desempenho das soluções propostas e implementadas. Abordamos em maiores detalhes: o ambiente implantado e configurado para realização dos experimentos; as ferramentas utilizadas na configuração dos enlaces e na composição de diferentes cenários de comunicação; a metodologia e planejamento dos experimentos; por fim, a análise de resultados a partir de indicadores de desempenho relacionados às métricas de QoS.

Tendo em vista que o objeto de estudo é desempenho das transmissões em multi-fluxos, o principal indicador de desempenho observado e analisado foi vazão da carga útil de dados, i.e. o *goodput*, o qual foi utilizado para comparar quantitativamente as soluções propostas com as existentes. Os experimentos foram planejados levando em consideração cenários comuns na Internet, de transmissão de dados tanto em rede cabeada quanto em rede sem fio. Isso nos possibilitou avaliar as soluções propostas sobre diversos tipos de situações que o MPTCP pode enfrentar em transmissões reais na Internet com ampla diversidade de multi-fluxos.

5.1 Ambiente Experimental

5.1.1 Topologia de Rede

O ambiente experimental implantado consiste em dois nós MPTCP, um cliente e um servidor, cada um com cinco interfaces de rede conectadas ponto-a-ponto, como ilustra a Figura 8 e descreve a Tabela 1. Para formar as cinco redes físicas distintas, em cada um dos nós foi utilizado a interface *Gigabit Ethernet* integrada na própria placa-mãe e uma placa adicional de rede *quad-port*, com quatro portas `giga ethernet`[1].

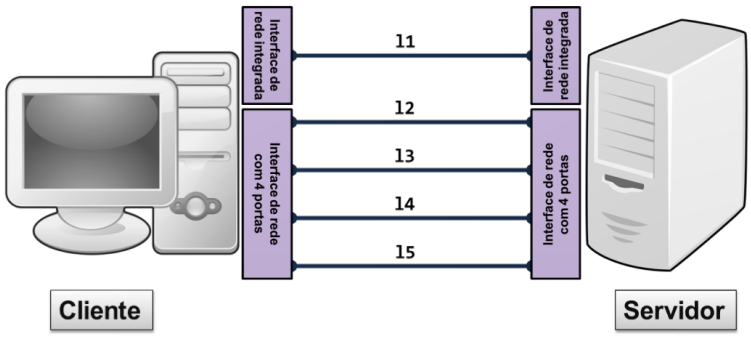


Figura 8 – Ambiente do experimento e sua topologia.

[1] Essas placas foram emprestadas de projeto número #2015/18808-0, financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), o qual é coordenado no ICT/Unifesp pelo orientador desta dissertação.

Tabela 1 – Componentes de hardware e software do ambiente experimental

Componente	Descrição
Cliente, Servidor	HP Proliant ML110 G4, Processador Intel Xeon Dual Core 2.3 GHz, 4 GB RAM, 500 GB HD
Interface Integrada	NIC single-port (`eth0`) Broadcomm 5721 Gigabit Ethernet, interface integrada na placa-mãe
Interface com 4 portas	NIC quad-port (`eth1`, `eth2`, `eth3`, `eth4`) Adaptador HP 331T Quad-Port Gigabit Ethernet
Número de caminhos	5 enlaces Gbps Ethernet (l_1, l_2, l_3, l_4, l_5)
Sistema Operacional	Ubuntu 15.04
MultiPath TCP	Versão 0.90, Linux kernel 3.18.26 - UCLouvain MPTCP Linux Kernel Implementation
Controle Congestionamento	LIA, OLIA, BALIA, wVegas
Gerenciador de Caminhos	*Full-mesh*, *Single-mesh*
Número de Sub-fluxos	$\|\mathcal{R}\| = 25$ sub-fluxos em *Full-mesh*, $\|\mathcal{R}\| = 5$ sub-fluxos em *Single-mesh*
Aplicação Teste	Cliente-servidor TCP para transferência massiva de dados (*bulk transfer*)

As placas de rede *quad-port* são do modelo 331T do fabricante HP, controlador Broadcomm 5719 e possuem soquete PCI-Express x4, não sendo possível utilizá-las em placas-mãe de *desktops* mais convencionais, que geralmente possuem soquete PCI-Express x16 configurado em *hardware* para ser compatível apenas com placas de vídeo. Cada um dos nós é uma estação de trabalho HP modelo Proliant ML110 G4[2], contendo os soquetes compatíveis com as placas de rede em PCI-Express x4. As estações de trabalho possuem configurações de *hardware* e *software* idênticas: processador Intel Xeon Dual Core 2.3 GHz, 4 GB de memória RAM, 500 GB de disco rígido, placa de rede *on-board* Broadcomm 5721 *Gigabit Ethernet*.

Em ambos os nós foram instalados o sistema operacional Ubuntu versão 15.04. O Ubuntu é um SO de código fonte aberto baseado na distribuição Debian GNU/Linux. Além de ser frequentemente atualizado e estável, uma das principais características do Ubuntu é sua usabilidade e facilidade de instalação de pacotes *softwares* (FOUNDATION, 2010). Por essa razão, os mantenedores do MPTCP disponibilizam o suporte à instalação do MPTCP via gerenciador *apt-get*[3].

Embora o ambiente experimental seja disponibilizado sobre uma topologia minimalista de dois nós conectados ponto-a-ponto, sua proposta é a de possibilitar diversidade de transmissão multi-fluxo com suporte de um controlador e emulador de rede, permitindo parametrizar cada um dos 5 enlaces em termos de vazão, latência e perda de pacotes. Nesses níveis de configurações de enlaces, torna-se possível criar experimentos variados, caracterizando redes de excelentes a péssimas condições de comunicação fim-a-fim, de baixo a alto congestionamento, independente se a topologia de rede é ponto-a-ponto ou multi-salto. Excelentes ou péssimas

[2] Essas estações de trabalho foram emprestadas pelo Departamento de Serviços de Informática (DSI) da Unifei.

[3] *Apt-get* é uma ferramenta de manipulação de pacotes de *software* por linha comando disponível em sistemas Debian, a qual possibilita *download* e instalação direta e automaticamente a partir de um repositório.

condição de rede são determinadas pela parametrização escolhida, por exemplo, a partir de uma taxa baixa ou alta de perda de pacote, uma taxa alta ou baixa de vazão, e uma taxa baixa ou alta de latência, respectivamente. A sub-seção seguinte descreve as ferramentas utilizadas na parametrização dos enlaces.

5.1.2 Ferramentas de Parametrização dos Enlaces

Os enlaces não podem ser parametrizados diretamente através do *driver* do controlador de interface de rede. Para configurar os parâmetros de QoS desejados, foi utilizado o controlador `tc` (*traffic control*), ferramenta nativa em sistemas Linux. O `tc`, na prática, consiste em um conjunto de ferramentas que permite controlar o tráfego de uma dada interface de rede (BROWN, 2006), sendo possível configurar sua taxa de transmissão (`rate`), pico de transmissão (*peakrate*), unidade máxima de transmissão (MTU), latência de transmissão, perda de pacotes, entre outros.

Os parâmetros configurados nos permitiram criar um ambiente controlado de tal modo que pudéssemos analisar a eficiência das soluções propostas. Para configurar a capacidade de transmissão dos enlaces, foi utilizado regulador de tráfego TBF (*Token Bucket Filder*), cujo comando é indicado no Código 5.1, onde `<eth>` é a interface alvo e `<x>` corresponde ao valor escolhido para o parâmetro. O parâmetro `limit` é referente ao tamanho da fila de espera em *bytes*, `rate` é a taxa de transmissão da interface, `mtu` é a unidade máxima de *bytes* transmitidos por pacote, `peakrate` é o pico da taxa de transmissão e o `buffer` é o tamanho do *buffer* instantâneo em *bytes*.

Código 5.1 – Comando `tc` para limitar a vazão dos enlaces.

```
1 $ tc qdisc add dev <eth> root handle 1: tbf limit <x>kb rate <x>mbit \
2   mtu <x> peakrate <x> mbit buffer <x>kb
```

O Código 5.2 mostra similarmente como regular a latência, onde o parâmetro `latency` define o tempo que os pacotes devem esperar em fila. Isso permite representar o atraso de enfileiramento nos roteadores existentes no caminho fim-a-fim.

Código 5.2 – Comando `tc` para limitar a vazão e latência dos enlaces.

```
1 $ tc qdisc add dev <eth> root handle 1: tbf rate <x>mbit \
2   mtu <x> peakrate <x>mbit buffer <x>kb latency <x>ms
```

Um emulador de rede, *network emulation* (`netem`), é parte do `tc` que permite adicionar às transmissões de pacotes outros parâmetros de QoS, como atraso e perda de pacote. Esses parâmetros requerem ser emulados, uma vez que não podem ser configurados diretamente sobre a interface de rede, como a definição da taxa de transmissão com suporte de TBF. A emulação é uma ferramenta importante para o estudo de protocolos e aplicações de redes, permitindo

realizar experimentos do os mais variados reais cenários encontrados em redes de dados (JURGELIONIS et al., 2011).

O `tc` permite configurar diferentes tratamentos de tráfego através da definição de classes de filas (*classful qdisc*) sobre uma interface de rede. Cada classe adicionada, portanto, permite a definição de um tratamento de tráfego específico. As classes são identificadas em hierarquia, sendo configuradas em formato numérico de `x:y`, onde `x` é a identificação da classe raiz e `y` é a classe filha. Por exemplo, `1:10`.

Para adicionar as configurações de perda e atraso no `tc`, utilizamos o `netem`. Como já possuímos a configuração raiz com TBF, adicionamos apenas uma classe filha ligada à configuração raiz. O Código 5.3 descreve a hierarquia de comandos, onde `<eth>` é a interface alvo e `<y>` corresponde ao percentual de perda de pacote desejado. Nesse comando, o parâmetro `parent` indica qual a raiz que estamos configurando, e o parâmetro `handle` identifica outras classes que podem ser incluídas na hierarquia.

Código 5.3 – Comando `tc` com emulador `netem` para definir a taxa de perda de pacotes.

```
1 $ tc qdisc add dev <eth> parent 1:1 handle 10: netem loss 0%
2 $ tc qdisc change dev <eth> parent 1:1 handle 10: netem loss <y>%
```

Com o uso de controlador `tc` e emulador `netem` foi configurada cada uma das interfaces de rede do ambiente ilustrado na Figura 8, possibilitando criar cenários de comunicação variados para avaliar o desempenho das soluções propostas para o MPTCP.

5.1.3 Ferramentas de Captura e Análise de Tráfego

A avaliação de desempenho das soluções propostas para o MPTCP foi realizada mediante à captura de pacotes e a análise de estatística básica sobre o tráfego gerado. Para isso, foram capturados todos os pacotes transmitidos em multi-fluxos MPTCP durante os experimentos, de modo que os dados de controle dos cabeçalhos da pilha TCP/IP foram salvos em disco e, posteriormente, processados para extrair as estatísticas desejadas.

A captura dos pacotes foi realizada com a ferramenta `tcpdump`[4], o qual é capaz de capturar e interpretar o cabeçalho do pacote transmitido em formato binário (FUENTES; KAR, 2005). O cabeçalho de transmissões multi-fluxos somente é interpretado a partir da versão 4.6.2 ou superior do *tcpdump*. O Código 5.4 apresenta o comando utilizado para capturar pacotes durante os experimentos, onde: `<eth>` é a interface alvo; `-p` define a captura em modo não promíscuo; `-s96` define a quantidade de bytes iniciais do pacote a ser capturada, geralmente limitando apenas a captura do cabeçalho do pacote, descartando o *payload*; `-vvv` define que a saída seja verbosa, capturando todo o tipo de informação necessário para análise; `-w <arquivo>` é o arquivo que armazenará o *dump* contendo dados dos cabeçalhos dos pacotes capturados que

[4] A ferramenta *tcpdump* foi desenvolvida em 1990 no Lawrence Berkeley National Laboratory (FUENTES; KAR, 2005).

Código 5.4 – Comando em `tcpdump` para capturar e salvar pacotes transmitidos em multi-fluxos durante os experimentos.

```
1   $ tcpdump.4.6.2 -p -s96 -i <eth> -vvv -w <arquivo> &
```

serão salvos; e `&` é para que o comando seja executado em segundo plano no terminal, sem que bloqueie o terminal para a execução de outros comandos. O comando deve ser executado para cada interface de rede.

A visualização dos pacotes capturados em formato binário é realizada pela ferramenta Wireshark. Essa ferramenta é um analisador de pacotes, cujo objetivo é mostrar em formato legível (alfa-numérico) a maior quantidade possível de dados sobre os pacotes trafegados na rede (LAMPING; SHARPE; WARNICKE, 2014). Wireshark permite a análise em tempo real de transmissão ou posterior, a partir de arquivos de *dumps*. Além de fornecer algumas estatísticas sobre os pacotes, Wireshark permite definir filtros para apresentação apenas de dados de pacotes de interesse.

Ao passo que Wireshark é uma ferramenta baseada em interface gráfica, o analisador Tshark permite executar funcionalidades do Wireshark em terminal (LAMPING; SHARPE; WARNICKE, 2014), possibilitando automatizar a análise e a extração de estatísticas desejadas sobre os pacotes capturados durante os experimentos. Nesse caso, com Tshark, basta definir os parâmetros de filtragem com linha de comando em terminal para, por exemplo, extrair métricas de QoS por segundo de transmissão, como latência, taxa de erro, vazão, entre outros.

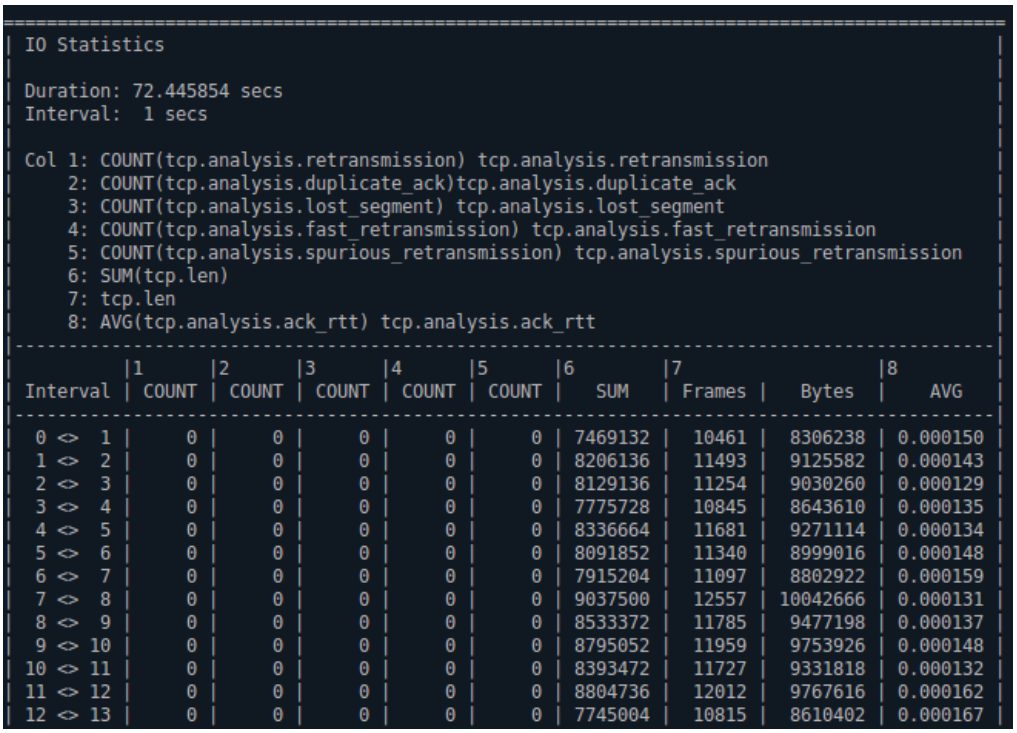
Para obter dados de desempenho do MPTCP a partir dos pacotes capturados durante as transmissões em multi-fluxos TCP, foi utilizado o comando descrito no Código 5.5. Os parâmetros foram configurados para analisar pacotes em intervalos de 1 segundo e extrair, respectivamente, os indicadores de: número de pacotes retransmitidos; número de pacotes `ACK` duplicados; número de pacotes perdidos; número de pacotes em estado de retransmissão rápida; número de pacotes espúrios retransmitidos; total de bytes transmitidos; total de bytes de carga útil transmitidos; média do RTT.

Código 5.5 – Comando do `tshark` para analisar e extrair informações dos arquivos *dumps* referentes às transmissões em multi-fluxos TCP.

```
1  $ tshark -r <arquivo_dump> -q -z io,stat,1,\
2  "COUNT(tcp.analysis.retransmission)tcp.analysis.retransmission",\
3  "COUNT(tcp.analysis.duplicate_ack)tcp.analysis.duplicate_ack",\
4  "COUNT(tcp.analysis.lost_segment)tcp.analysis.lost_segment",\
5  "COUNT(tcp.analysis.fast_retransmission)tcp.analysis.fast_retransmission",\
6  "COUNT(tcp.analysis.spurious_retransmission)tcp.analysis.
       spurious_retransmission",\
7  "SUM(tcp.len), tcp.len", "AVG(tcp.analysis.ack_rtt)tcp.analysis.ack_rtt"
```

A ferramenta Tshark retorna as análises em tabelas, como ilustra a Figura 9. Os dados

extraídos são novamente armazenados, desta vez em um arquivo texto para posterior análise estatística e geração dos gráficos resultantes.



```
================================================================================
| IO Statistics                                                                |
|                                                                              |
| Duration: 72.445854 secs                                                     |
| Interval:  1 secs                                                            |
|                                                                              |
| Col 1: COUNT(tcp.analysis.retransmission) tcp.analysis.retransmission        |
|     2: COUNT(tcp.analysis.duplicate_ack)tcp.analysis.duplicate_ack           |
|     3: COUNT(tcp.analysis.lost_segment) tcp.analysis.lost_segment            |
|     4: COUNT(tcp.analysis.fast_retransmission) tcp.analysis.fast_retransmission |
|     5: COUNT(tcp.analysis.spurious_retransmission) tcp.analysis.spurious_retransmission |
|     6: SUM(tcp.len)                                                          |
|     7: tcp.len                                                               |
|     8: AVG(tcp.analysis.ack_rtt) tcp.analysis.ack_rtt                        |
|------------------------------------------------------------------------------|
|           |1     |2     |3     |4     |5     |6       |7                |8       |
| Interval  | COUNT | COUNT | COUNT | COUNT | COUNT |   SUM   | Frames |   Bytes  |    AVG   |
|------------------------------------------------------------------------------|
|  0 <>  1  |     0 |     0 |     0 |     0 |     0 | 7469132 |  10461 |  8306238 | 0.000150 |
|  1 <>  2  |     0 |     0 |     0 |     0 |     0 | 8206136 |  11493 |  9125582 | 0.000143 |
|  2 <>  3  |     0 |     0 |     0 |     0 |     0 | 8129136 |  11254 |  9030260 | 0.000129 |
|  3 <>  4  |     0 |     0 |     0 |     0 |     0 | 7775728 |  10845 |  8643610 | 0.000135 |
|  4 <>  5  |     0 |     0 |     0 |     0 |     0 | 8336664 |  11681 |  9271114 | 0.000134 |
|  5 <>  6  |     0 |     0 |     0 |     0 |     0 | 8091852 |  11340 |  8999016 | 0.000148 |
|  6 <>  7  |     0 |     0 |     0 |     0 |     0 | 7915204 |  11097 |  8802922 | 0.000159 |
|  7 <>  8  |     0 |     0 |     0 |     0 |     0 | 9037500 |  12557 | 10042666 | 0.000131 |
|  8 <>  9  |     0 |     0 |     0 |     0 |     0 | 8533372 |  11785 |  9477198 | 0.000137 |
|  9 <> 10  |     0 |     0 |     0 |     0 |     0 | 8795052 |  11959 |  9753926 | 0.000148 |
| 10 <> 11  |     0 |     0 |     0 |     0 |     0 | 8393472 |  11727 |  9331818 | 0.000132 |
| 11 <> 12  |     0 |     0 |     0 |     0 |     0 | 8804736 |  12012 |  9767616 | 0.000162 |
| 12 <> 13  |     0 |     0 |     0 |     0 |     0 | 7745004 |  10815 |  8610402 | 0.000167 |
```

Figura 9 – Exemplo de saída da ferramenta Tshark.

Para obtenção da vazão com saída formatada em *bytes*/s e *megabytes*/s, a ferramenta Tshark e outra ferramentas de processamento de texto nativa em ambiente Linux, como `sed` e `awk`, podem ser combinadas, como ilustra do Código 5.6.

Código 5.6 – Comando do `tshark` para obter a taxa de transmissão em *bytes*/s e *megabytes*/s.

```
1 $ tshark -r <arquivo_dump> -T fields -e frame.time -e frame.len \
2 | sed -e 's/\..*\t/\t/' | awk -F"\t" '$1==last {sum += $2; next} \
3 {printf("%8d bytes/s (%6.2f Mbit/s)\n",sum,sum*8/1024/1024);last=$1;sum=$2}
    '
```

Outra ferramenta útil é a `tcpstat`, que também é capaz de analisar o arquivo *dump* e extrair indicadores de desempenho desejados, como vazão utilizada, número de pacotes, tamanho médio de pacotes, entre outros (HERMAN, 2001). Um exemplo de comando em `tcpstat` é mostrado no Código 5.7, onde o `<arquivo_dump>` é o arquivo *dump* e `<numero_porta>` é o número da porta da aplicação alvo.

Código 5.7 – Comando do `tcpstat` para obter a taxa de transmissão em *bytes*/s

```
1 $ tcpstat -r <arquivo_dump> -f "port <numero_porta>" -o '%B\n' > <
    arquivo_vazao>
```

5.1.4 Aplicação de Geração de Tráfego

Para avaliar o desempenho das transmissões em multi-fluxos, foi implementada na linguagem C uma aplicação do tipo cliente-servidor TCP para *upload* de conteúdo com transmissão em massa (*TCP bulk transfer*). Para tanto, a aplicação no lado do servidor operou sobre porta efêmera (9000) para estabelecimento de conexão TCP, recebendo mensagens de 3 MB enviadas pelo cliente.

Para estressar a operação do servidor e gerar transmissão contínua e em massa, o cliente enviou um total de 500 mensagens ininterruptas. Portanto, o volume total de *bytes* transmitidos por execução do cliente foi de 1.5 GB. O servidor, mesmo após receber todas as mensagens, foi implementado para retornar ao ponto de espera (`accept`) de uma nova conexão do cliente.

5.1.5 Configuração do Protocolo MPTCP

No MPTCP, o sistema final emissor é o que executa os principais mecanismos que determinam a capacidade de transmissão em multi-fluxo. Assim, o gerenciador de caminhos (*path manager*), o escalonador de sub-fluxo e o controle de congestionamento foram configurados e executados no emissor cliente.

Para realizar a troca dos algoritmos de controle de congestionamento em tempo de execução do sistema operacional, foi necessário instalar os respectivos módulos. O comando utilizado para isso é apresentado no Código 5.8, onde `<caminho_modulo>` é o caminho do arquivo de módulo a ser instalado.

Código 5.8 – Comando para instalação de módulo do Kenel Linux em tempo de execução.

```
1 $ insmod <caminho_modulo>
```

O algoritmo LIA é utilizado como controle de congestionamento padrão do MPTCP, como introduzido na Seção 3.1. Portanto, o módulo LIA é pré-carregado automaticamente pelo kernel Linux. Para avaliar as soluções propostas sobre diferentes algoritmos de controle de congestionamento, foi necessário realizar a instalação de outros três módulos de controle de congestionamento: OLIA, BALIA e wVegas.

Uma vez instalados, o comando em terminal para trocar o algoritmo é indicado no Código 5.9, onde o `<nome_algoritmo>` é o algoritmo de controle de congestionamento alvo (`lia`, `olia`, `balia`, `wvegas`).

Código 5.9 – Comando para troca do algoritmo do controle de congestionamento.

```
1 $ sysctl -w net.ipv4.tcp_congestion_control=<nome_algoritmo>
```

Assim como os escalonadores nativos do MPTCP, as soluções que propomos estão implementados de forma modular no protocolo, no núcleo do SO. Ao recompilar o SO com os escalonadores propostos, pode-se configurar a política de escalonamento desejada em tempo

de execução do SO. O comando de terminal para essa configuração é indicado pelo Código 5.10, em `<nome_escalonador>`, tendo as opções dos escalonadores: nativos do MPTCP, LL (`default`), Round-Robin (`roundrobin`), RDR (`redundant`); e as soluções que propomos, *Largest Window Space* (`lws`), *Lowest Time/Space* (`lts`), *Highest Sending Rate* (`hsr`).

Código 5.10 – Comando para troca do algoritmo de escalonamento de sub-fluxos.

```
1 $ sysctl -w net.mptcp.mptcp_scheduler=<nome_escalonador>
```

5.2 Avaliação das soluções sobre caminhos congestionados

Na primeira bateria de experimentos, o objetivo foi avaliar o desempenho das soluções propostas para escalonamento de pacotes, considerando cenários de comunicação multi-fluxos sobre enlaces de capacidade heterogênea, em diferentes taxas de perda de pacote.

5.2.1 Metodologia

Um que vez que o congestionamento no sub-fluxo é sentido no sistema final emissor através da detecção da perda dos pacotes enviados em janela w_r, ao variar as taxas de perda de pacotes permite-se introduzir nos experimentos níveis de congestionamentos mais ou menos agressivos nos sub-fluxos. Nesses cenários, identificamos degradação de desempenho do escalonador LL, conforme problema descrito na Seção 1.1.1. Portanto, são cenários úteis para comparação das soluções propostas, utilizando LL como *baseline*.

Uma aplicação auxiliar implementada no cliente foi responsável por realizar a configuração; i) da capacidade dos enlaces das cinco interfaces de rede em 10, 25, 50, 75 e 100 Mbps, e ii) da emulação da perda de pacotes a cada rodada de experimentos. Nessa bateria de experimentos, a taxa de perda de pacotes foi variada de 0 a 12 % e em modo uniforme nas cinco interfaces de rede, ou seja, em cada experimento, as cinco interfaces foram configuradas com mesma taxa de erro. As taxas foram limitadas a 12 %, pois observamos que em taxas acima desse percentual as vazões dos sub-fluxos convergem entre si, não tendo diferença de desempenho, independentemente do escalonador e da capacidade de transmissão dos caminhos fim-a-fim. Ao configurar a emulação de modo uniforme, com mesma taxa de perda de pacotes sobre as cinco interfaces, permite-se representar cenários frequentes de comunicações reais, onde os caminhos existentes a partir da rede de origem passam por um mesmo enlace de gargalo na rede para chegar ao destino. Nesses cenários, todos os caminhos do MPTCP emissor estarão sujeitos aos ruídos e congestionamento de um mesmo enlace de gargalo, apresentando uma taxa similar de perda de pacotes conforme o estado atual desse enlace.

O total de experimentos realizados nessa bateria é definido por

$$N_{te} = n_{es} \times n_{cc} \times n_{te}, \tag{5.1}$$

onde n_{es} é o número de escalonadores observados, n_{cc} é o número de algoritmos de controle de congestionamento observados e n_{te} é a quantidade de taxas de perda de pacotes configuradas.

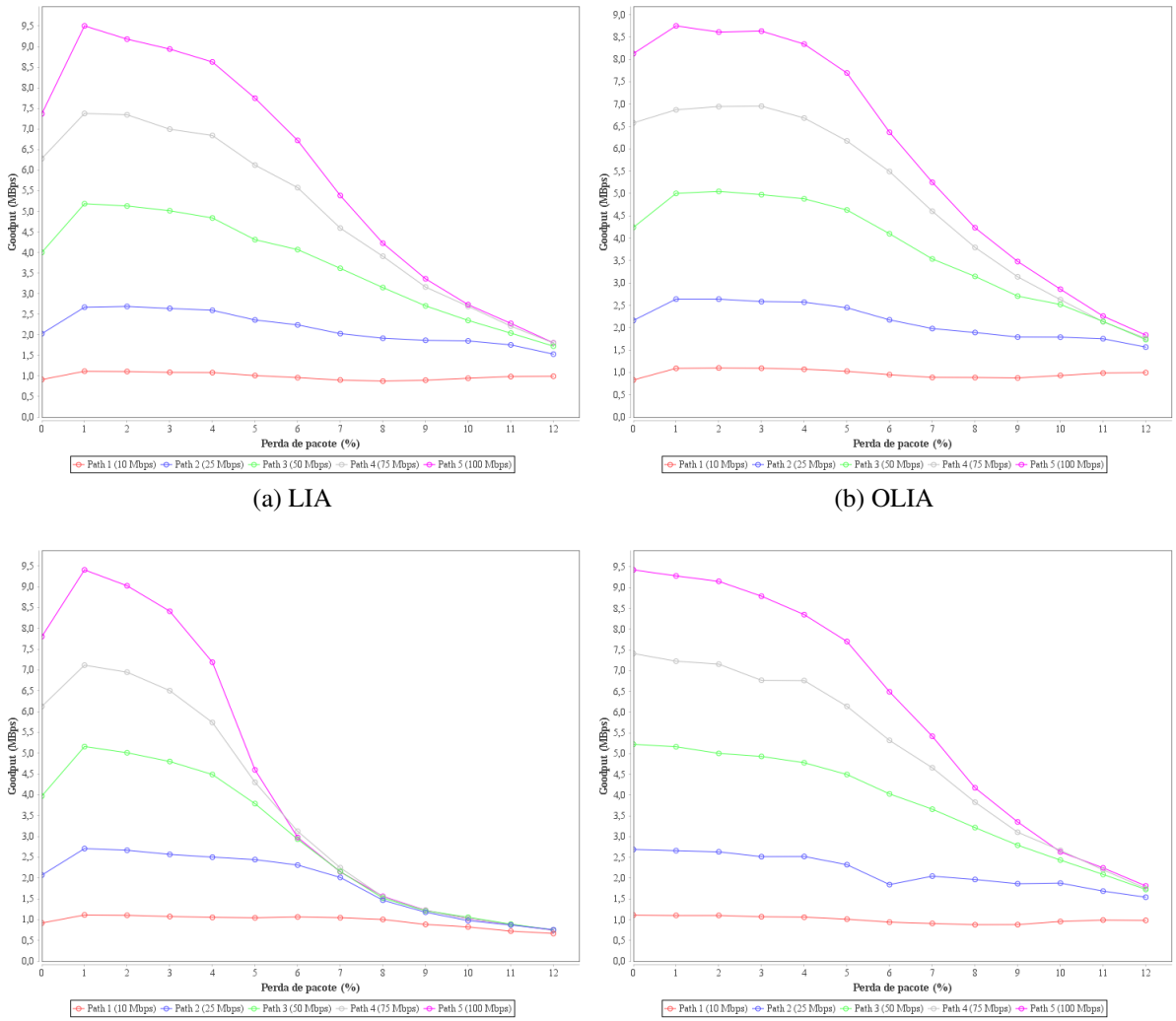
Nessa primeira bateria, o total de $N_{te} = 208$ experimentos foram realizados, assumindo: $n_{es} = 4$, considerando o escalonador padrão LL como *baseline* de comparação para os outros três escalonadores propostos, LWS, LTS, HSR; $n_{cc} = 4$, considerando os quatro algoritmos de controle de congestionamento existentes, LIA, OLIA, BALIA e wVegas; e $n_{te} = 13$, considerando a variação da taxa de erro dos enlaces de 0 % a 12 %. Portanto, cada solução de escalonamento foi avaliada sobre todos os quatro algoritmos de congestionamento e em todas as treze taxas de perda de pacotes configuradas.

A captura dos pacotes sobre as cinco interfaces de rede foi realizada com a ferramenta `tcpdump` no nó emissor cliente, como explicado na Seção 5.1.3. Os arquivos *dumps* de cada interface de rede foram salvos separadamente, sendo organizados hierarquicamente por solução de escalonamento, algoritmo de controle de congestionamento e taxa de erro observada. Os resultados são discutidos nas sub-seções seguintes para cada um dos escalonadores.

5.2.2 Escalonador LL

Na análise de desempenho do escalonador LL os gráficos resultantes apresentam a vazão obtida em cada caminho (interface de rede), conforme apresenta a Figura 10. Importante observar que o gerenciador de caminhos utilizado nessa bateria de experimentos foi *Full-mesh*, configuração padrão do MPTCP. Nesse caso, cada caminho foi compartilhado por cinco sub-fluxos.

Nos gráficos das Figuras 10(a), 10(b) e 10(c), podemos notar um aumento na vazão da taxa de perda de 0 % para 1 % nos experimentos com os algoritmos de controle de congestionamento LIA, OLIA e BALIA, respectivamente. O baixo desempenho quando os caminhos não estão congestionados (em perda 0 %) ocorre devido à concorrência causada entre os próprios sub-fluxos estabelecidos pelo gerenciador *Full-mesh*, o qual cria sub-fluxos para estabelecer conexões entre todas as combinações de endereços entre dois sistemas finais. Em nosso cenário experimental, temos 25 sub-fluxos, considerando os endereçamentos das 5 interfaces de rede em cada nó. Assim, há pelo menos 5 sub-fluxos concorrendo pelo uso do enlace disponível em cada caminho. Como os algoritmos de controle de congestionamento LIA, OLIA e BALIA inferem congestionamento com base no evento de perda de pacote, a diminuição multiplicativa das janelas dos sub-fluxos em $w_r/2$ desses algoritmos se torna agressiva, não sendo responsivo de forma eficiente ao congestionamento gerado pelos próprios sub-fluxos que compartilhavam o mesmo enlace. O efeito é um impacto global na diminuição da vazão de todos sub-fluxos.



(a) LIA (b) OLIA

(c) BALIA (d) wVegas

Figura 10 – Vazão total multi-fluxo obtida com o escalonador padrão LL, utilizando o gerenciador de caminhos *Full-mesh* e diferentes algoritmos de controle de congestionamento.

No algoritmo de congestionamento wVegas, conforme resultados apresentados no gráfico da Figura 10(d), podemos observar que esse efeito não ocorre. A vazão se mantém equilibrada quando a rede está em estado de baixo congestionamento, em perdas de 0 a 3 % de pacotes. Isso ocorre devido à operação de wVegas que, não somente se baseia somente no evento de perda, mas também regula as janelas w_r dos sub-fluxos com maior sensibilidade através da transmissão e o atraso de enfileiramento atual dos enlaces. Como resultado, ocorre a drenagem dos sub-fluxos mais congestionados e melhor distribuição do tráfego dos pacotes entre os caminhos existentes, consequentemente, melhor desempenho.

O controle de congestionamento BALIA, como mostra o gráfico da Figura 10(c), degrada consideravelmente o desempenho da transmissão conforme o aumento da taxa de perda de pacotes. Isso ocorre devido ao objetivo primário do algoritmo, de prover um bom balanceamento entre responsividade e justiça/amistosidade no uso de enlaces compartilhados com

conexões TCP de caminhos únicos. O resultado é uma retração da janela mais agressiva das janelas w_r, conforme o congestionamento (taxa de perda) aumenta.

Nesse experimento pudemos verificar que a perda de pacote degrada muito o desempenho da vazão. O fato de o LL com o uso de estimativas sRTT não poder trazer informações sobre caminhos ruidosos leva a degradação do desempenho da vazão de dados. Por exemplo, considere dois sub-fluxos: um sub-fluxo a com de menor latência, porém com maior perda de pacote; e um sub-fluxo b, com latência maior, mas menor taxa de perda de pacotes. O escalonador LL escolherá o sub-fluxo a, de menor latência, ocasionando maior perdas de pacotes.

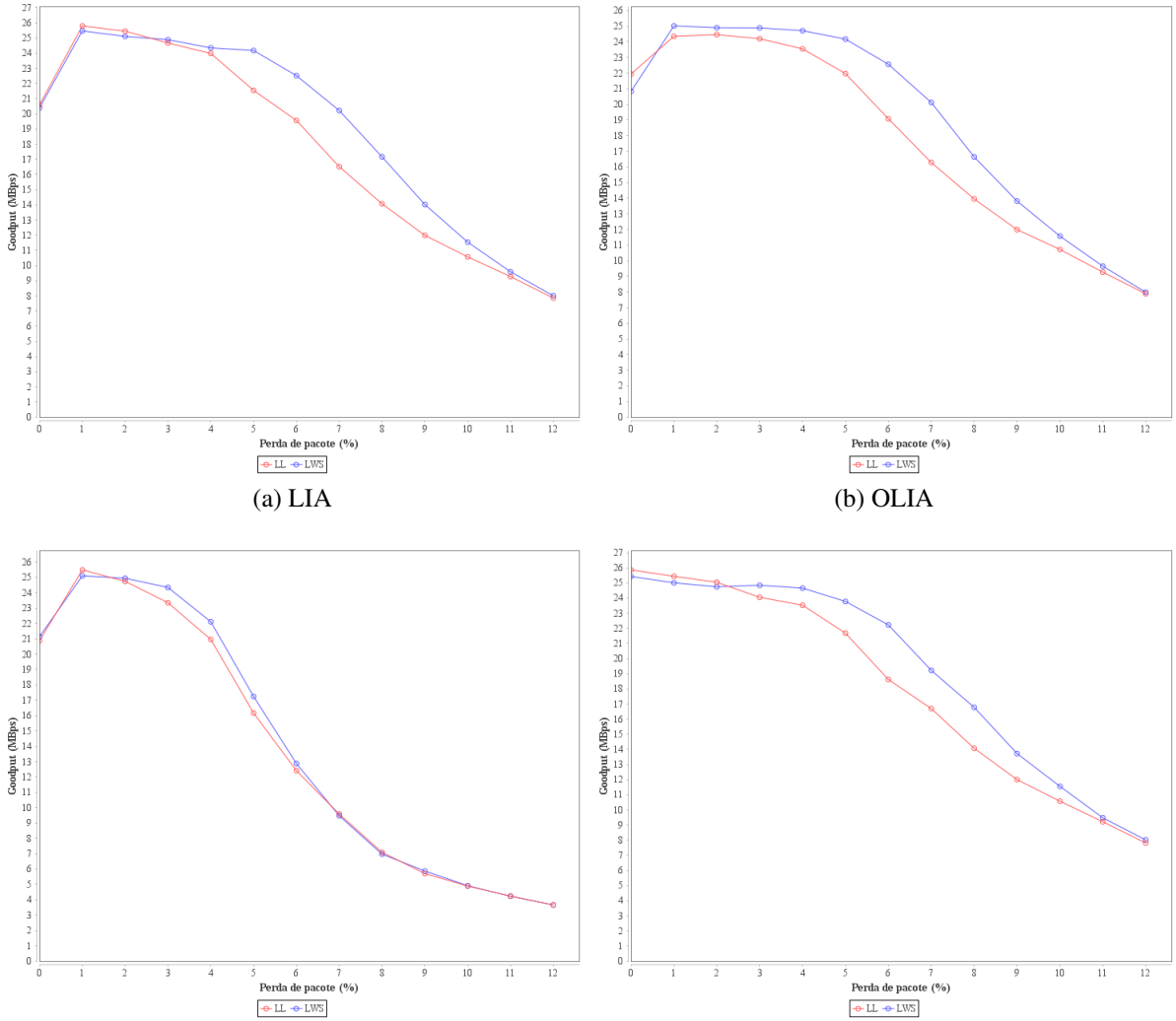
5.2.3 Escalonador LWS

Nos gráficos da Figura 11, entre as taxas de perda de 4 % a 12 %, é observado um ganho de desempenho considerável do LWS em relação ao LL sobre diferentes algoritmos de controle de congestionamento, como LIA, OLIA e wVegas. Tal resultado se deve ao fato de o LWS conseguir diferenciar os melhores sub-fluxos através do maior espaço na janela, principalmente cenários de maior congestionamento, como observou nas taxas de perdas de pacote superiores à 4 %. A hipótese levantada de escolha inconsistente do algoritmo LL é observada experimentalmente em cenários de maior congestionamento em todos os algoritmos de controle de congestionamento, os quais são refletidos em maiores taxas de perda de pacotes, onde o algoritmo proposto LWS teve melhor desempenho.

Pudemos observar que o escalonador de sub-fluxos influencia diretamente o desempenho, dependendo do parâmetro escolhido para escalonamento. No entanto, o ganho na vazão depende também do controle de congestionamento. Enquanto LWS prioriza os sub-fluxos de maior espaço na janela, a expansão e retração das janelas são determinadas pelos respectivos algoritmos de controle de congestionamento. Isso é possível observar através dos resultados de desempenho entre os algoritmos de controle de congestionamento. Enquanto wVegas provê o melhor desempenho por não depender exclusivamente do evento de perda para regular as janelas dos sub-fluxos, o algoritmo BALIA teve a menor vazão multi-fluxo, com ganho de LWS não muito expressivo sobre LL, conforme apresenta o gráfico da Figura 11(c). Na tentativa de prover um bom balanceamento entre reponsividade e justiça, BALIA apresentou pior desempenho dos sub-fluxos, penalizando a vazão dos sub-fluxos, tanto em relação ao escalonadores quanto em relação aos outros controles de congestionamento.

5.2.4 Escalonador LTS

A comparação entre os escalonadores LTS e LL é apresentada nos gráficos da Figura 12. Assim como LWS, entre as taxas de perda de 4% a 10%, o ganho de vazão é superior em relação ao LL, principalmente sobre os algoritmos de controle de congestionamento LIA, OLIA e wVegas. Tal resultado se deve a escolha de sub-fluxos que possuem a menor relação



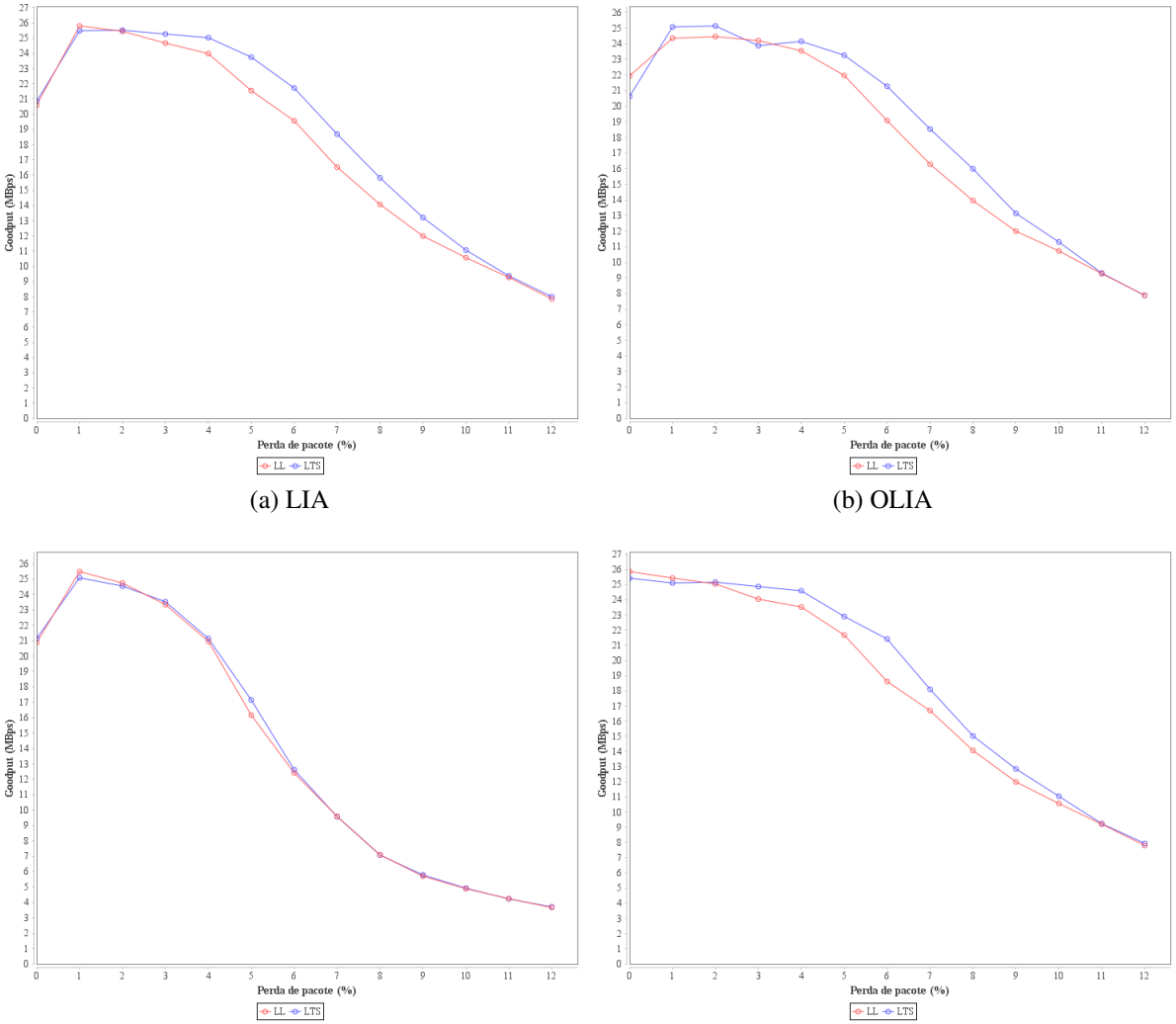
(a) LIA (b) OLIA

(c) BALIA (d) wVegas

Figura 11 – Vazão total multi-fluxo obtida com o escalonador LWS em comparação ao escalonador padrão LL, utilizando o gerenciador de caminhos *Full-mesh* e diferentes algoritmos de controle de congestionamento.

entre latência e espaço na janela, o LTS obteve neste cenário um desempenho igual ou superior que o LL, principalmente para taxas de perda de pacote superiores à 4 %. Tal resultado é devido ao fato de o escalonador LTS priorizar os sub-fluxos com menor relação entre latência e espaço de janela, o que leva a escolha de melhores sub-fluxos com menor latência e, ao mesmo tempo, de maior espaço na janela.

O algoritmo BALIA novamente apresentou pior desempenho, como mostra gráfico na Figura 12(c). Observamos que sobre BALIA, o ganho de desempenho de LTS foi praticamente nulo, comparado ao LL. Nesse caso, LTS foi mais sensível ao controle de janela em BALIA, o qual diminui a janela de forma mais agressiva para que os multi-fluxos MTCP sejam mais amigáveis com as conexões de TCP comum.



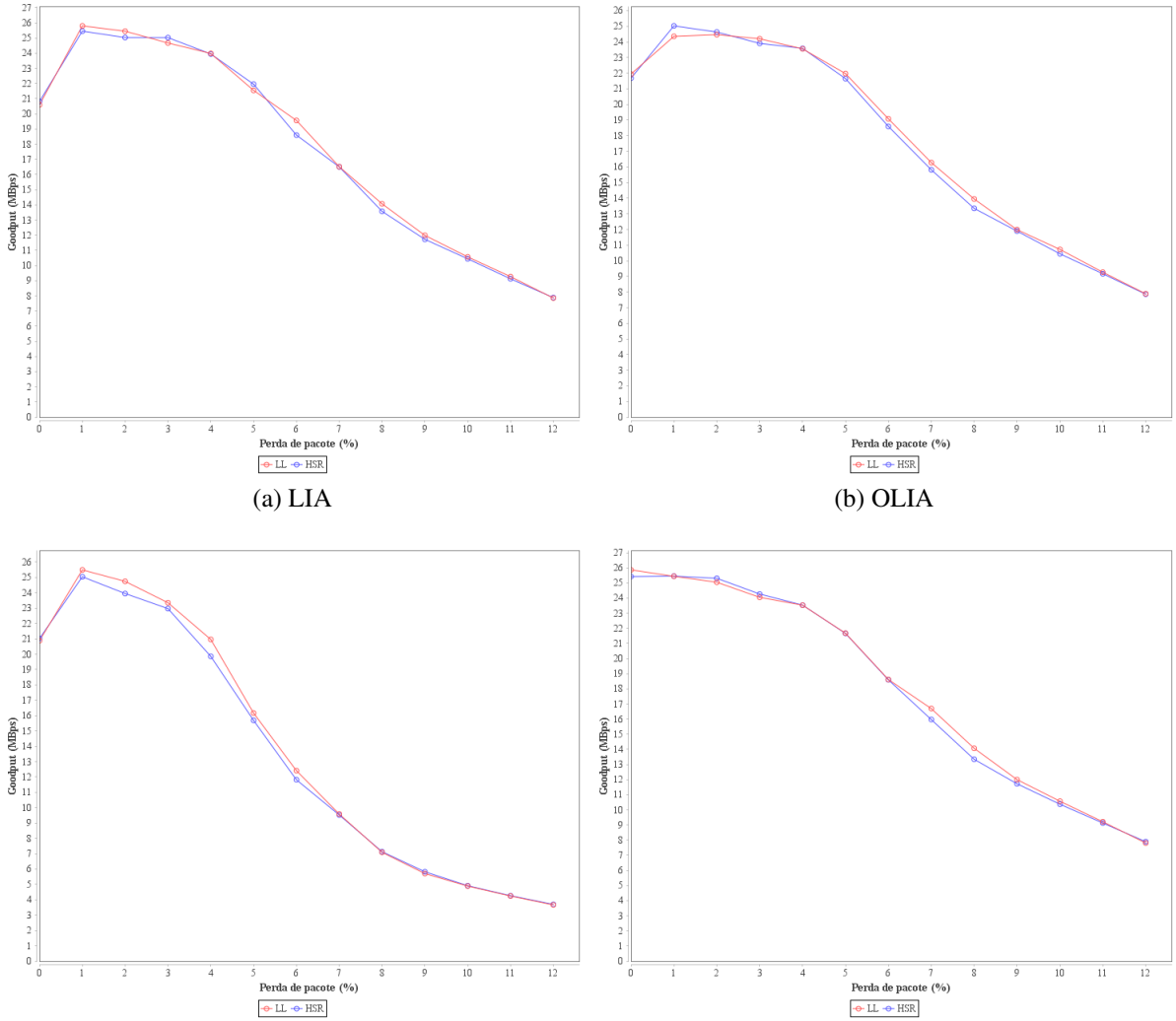
(a) LIA (b) OLIA

(c) BALIA (d) wVegas

Figura 12 – Vazão total multi-fluxo obtida com o escalonador LTS em comparação ao escalonador padrão LL sobre o gerenciador de caminhos *Full-mesh* em diferentes algoritmos de controle de congestionamento.

5.2.5 Escalonador HSR

Os resultados obtidos dos experimentos realizados com LL e HSR são apresentados nos gráficos da Figura 13. De modo geral, HSR apresentou um desempenho bastante similar ao LL. Retomando a definição de taxa de envio, sendo $sr = w_r \times m_r/\ddot{\tau}_r$, o desempenho próximo de LL ocorre, pois: $i)$ ao passo que o numerador é determinado por valores baixos, uma vez que as janelas w_r se mantêm retraídas em estado de perda devido à operação dos algoritmos de controle de congestionamento; $ii)$ o denominador é definido pela medição instantânea de RTT. Assim, sob estado de perda, o uso da latência como denominador, mesmo sendo a medição instantânea, gera praticamente o mesmo efeito que o escalonamento em LL com a estimativa em sRTT.



(a) LIA (b) OLIA

(c) BALIA (d) wVegas

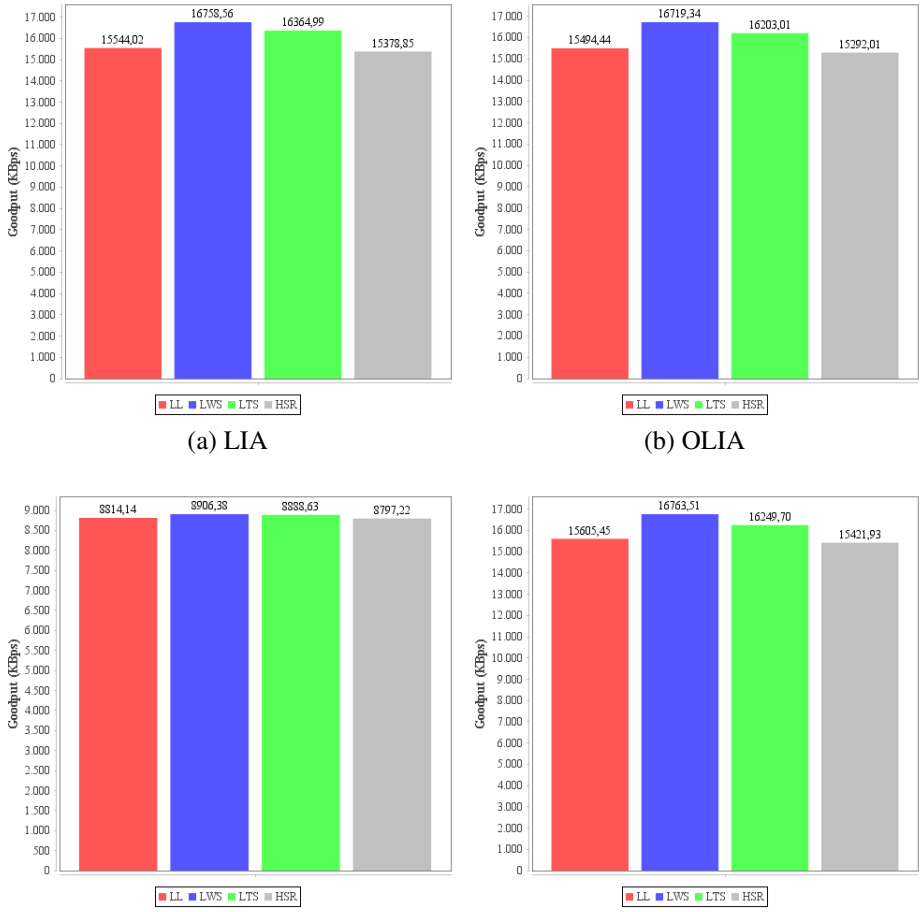
Figura 13 – Vazão total multi-fluxo obtida com o escalonador HSR em comparação ao escalonador padrão LL sobre o gerenciador de caminhos *Full-mesh* em diferentes algoritmos de controle de congestionamento.

5.2.6 Discussão

Para uma comparação geral dos resultados obtidos dos escalonadores, analisamos aqui a vazão de todos os experimentos sobre cada controle de congestionamento. As Figuras 14 e 15 apresentam essas comparações entre os escalonadores através de gráficos de barras e *boxplots*.

Conforme os resultados apresentados nos gráficos, observamos o desempenho notório do LWS entre todos os escalonadores. Esse resultado valida a Hipótese 1, onde os sub-fluxos mais rápidos tendem a ter mais espaço na janela por sua rápida transmissão mesmo em cenários de transmissão ruidosas, com maior taxa de perda de pacotes.

Uma observação importante entre os controles de congestionamento é o impacto do controle de congestionamento BALIA. Os resultados simulares entre os escalonadores se deve à estratégia do algoritmo do BALIA, em que um dos principais objetivos é possibilitar um bom



(a) LIA (b) OLIA

(c) BALIA (d) wVegas

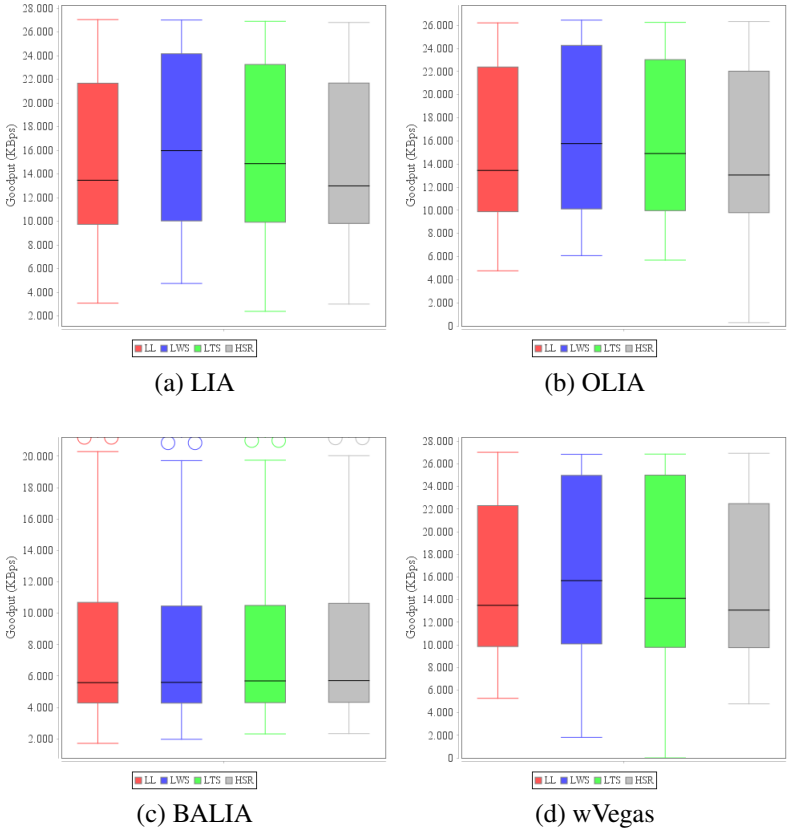
Figura 14 – Média da vazão multi-fluxo obtida com o escalonadores LL, LWS, LTS e HSR sobre o gerenciador de caminhos *Full-mesh* em diferentes algoritmos de controle de congestionamento.

balanceamento entre responsividade e amistosidade no compartilhamento de enlaces com conexões de TCP, de fluxos únicos. Para tanto, há uma menor agressividade na expansão das janelas dos sub-fluxos, na fase de prevenção de congestionamento, e uma maior agressividade na retração das janelas, ao detectar congestionamento pelo evento de perda de pacotes. O impacto de BALIA no desempenho acaba sendo uma vazão na transmissão multi-fluxo consideravelmente menor, independente da política de escalonamento.

Observamos que o desempenho de LTS está entre o de LL e LWS. Isso se deve ao fato de LTS aplicar um comparador composto, a partir da junção dos parâmetros τ_r (menor sRTT do comparador do LL) e o espaço ws existente nas janelas (comparador do LWS).

O escalonador HSR foi pior no geral para todas as soluções, tal razão se deve a capacidade estimada ser uma informação imprecisa sobre cenários ruidosos, entretanto conforme vemos nos experimentos seguintes, o HSR obtêm resultados positivos.

Nessa metodologia experimental, variamos de forma uniforme a taxa de perda de pacotes dos caminhos para representar cenários onde os sub-fluxos de capacidades heterogêneas



(a) LIA (b) OLIA

(c) BALIA (d) wVegas

Figura 15 – *Boxplot* da vazão multi-fluxo obtida com o escalonadores LL, LWS, LTS e HSR sobre o gerenciador de caminhos *Full-mesh* em diferentes algoritmos de controle de congestionamento.

compartilham um mesmo enlace congestionado. Observamos que o LWS conseguiu escalonar os sub-fluxos de modo eficiente, tal que as capacidades dos caminhos foram melhor exploradas, mesmo sob maiores taxas de perdas em enlaces ruidosos ou congestionados. Pudemos verificar que o escalonador padrão LL sofre mais em cenários de comunicação com essas características, não balanceando de forma mais eficiente a carga sobre os sub-fluxos.

5.3 Avaliação das soluções sobre caminhos homogêneos e heterogêneos

Nesta análise utilizamos a mesmas ferramentas descritas na Seção 5.1. Entretanto, os caminhos da topologia foram parametrizados de forma heterogênea e homogênea, considerando a taxa de transmissão, latência e a taxa de perda de pacote de cada um dos cinco enlaces.

5.3.1 Metodologia

Embora nossa topologia seja minimalista, nós definimos cenários com diversas características que podem ser encontradas num ambiente real, de tal modo que podemos avaliar as

implementações propostas em condições de rede semelhantes a da Internet. Assim como a rede experimental implantada por Paasch et al. (2014a), essa topologia de salto único possibilita identificar o limite de desempenho dos escalonares propostos em caminhos de diferentes condições de transmissão.

Para possibilitar diversidade multi-fluxo e identificar em quais condições de caminho um escalonador é pior ou melhor que outro, aplicamos a abordagem de planejamento de experimentos proposta por Paasch, Khalili e Bonaventure (2013). Assim, selecionamos os principais fatores que influenciam o desempenho do MPTCP e determinamos os seus domínios de valores através da parametrização homogênea (tipo ⓪) e heterogênea (tipo ❶) dos caminhos do ambiente experimental (Figura 8), conforme descritos nas Tabelas 2 e 3. Para reduzir o espaço de experimentos, mas mantendo cenários realistas com condições de rede que o MPTCP pode enfrentar na Internet, definimos intervalos discretos de valores de parametrição baseados em vários estudos de medições de transmissões com fio, conforme (BOVY et al., 2002), (ZHANG et al., 2010) e (NGUYEN; ROUGHAN, 2013), e transmissões sem fio, de acordo com (CHEN et al., 2013), (NIKRAVESH et al., 2016) e (BISWAS et al., 2015). Nesse contexto, na rede experimental ponto-a-ponto podem ser produzidas transmissões variadas, de ruins a excelentes, que seriam sentidas pelas variáveis de estado do MPTCP (e.g., w_r, τ_r, f_r) em redes de multi-saltos na Internet. Portanto, a diversidade de multi-fluxo necessária pelos escalonadores para a tomada de decisão é igualmente obtida através da configuração de salto-único.

Tabela 2 – Parametrização dos Caminhos Homogêneos

Tipo ⓪	**Domínio de valores para cada caminho**				
Fatores	l_1	l_2	l_3	l_4	l_5
Taxa de Perda	0%	0%	0%	0%	0%
Taxa de Transmissão	10 Mbps	10 Mbps	10 Mbps	10 Mbps	10 Mbps
Latência fim-a-fim	5 ms	5 ms	5 ms	5 ms	5 ms

Tabela 3 – Parametrização dos Caminhos Heterogêneos

Tipo ❶	**Domínio de valores para cada caminho**				
Fatores	l_1	l_2	l_3	l_4	l_5
Taxa de Perda	0%	1%	2%	6%	12%
Taxa de Transmissão	10 Mbps	25 Mbps	50 Mbps	75 Mbps	100 Mbps
Latência fim-a-fim	5 ms	10 ms	20 ms	40 ms	80 ms

Os experimentos foram planejados a partir da combinação dos três fatores e as duas possibilidades de configuração. Obtemos um total de oito experimentos, contemplando todos os possíveis cenários nessa combinação, conforme apresenta a Tabela 4.

O total de experimentos realizados é dado por

Tabela 4 – Experimentos planejados conforme as parametrizações homogênea e heterogênea dos caminhos.

	Parametrização dos caminhos		
Experimentos	**Taxa de Perda**	**Taxa de Trans.**	**Latência fim-a-fim**
Exp 1	⓪	⓪	⓪
Exp 2	⓪	⓪	❶
Exp 3	⓪	❶	⓪
Exp 4	⓪	❶	❶
Exp 5	❶	⓪	⓪
Exp 6	❶	⓪	❶
Exp 7	❶	❶	⓪
Exp 8	❶	❶	❶

$$N_{te} = |\text{Exp}| \times n_{es} \times n_{cc}, \tag{5.2}$$

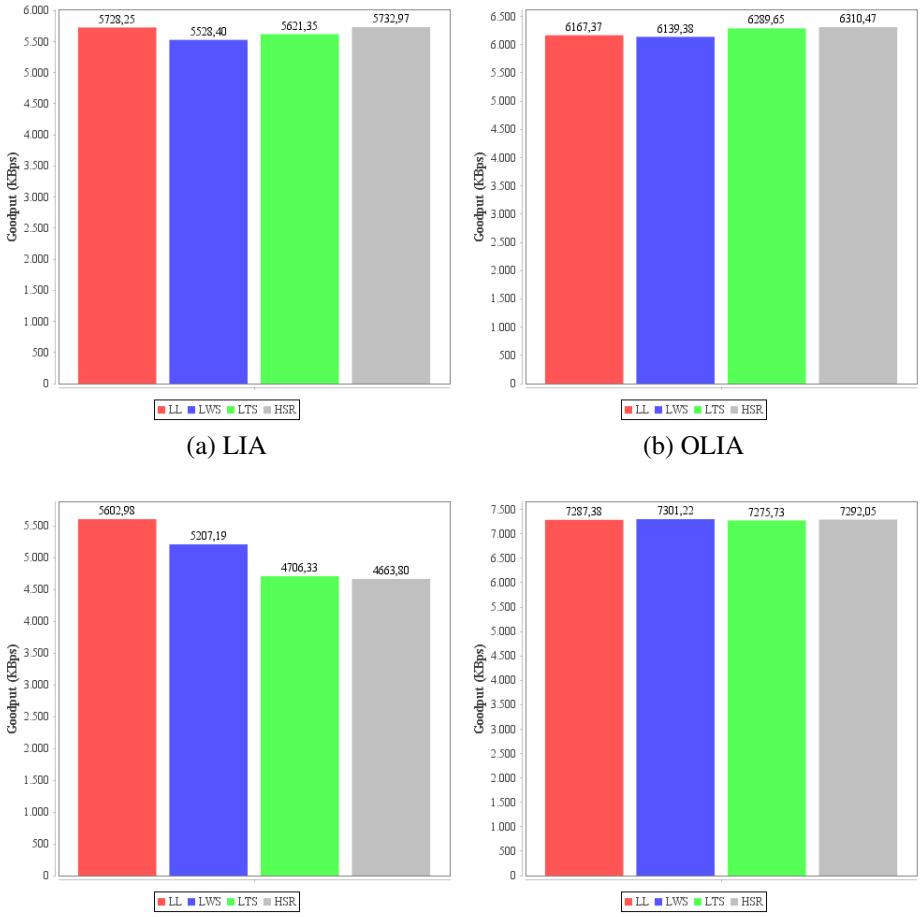
onde: $|\text{Exp}| = 2^3 = 8$ experimentos, considerando os dois tipos de domínios de valores (homogêneo ⓪ e heterogêneo ❶) de parametrização dos caminhos e os três fatores (taxa de transmissão, taxa de perda, latência fim-a-fim); $n_{es} = 4$, considerando o número de escalonadores observados (LL, HSR, LWS, LTS); $n_{cc=4}$, considerando número de algoritmos de controle de congestionamento observados (LIA, OLIA, BALIA, wVegas). Portanto, nesta segunda metodologia experimental, realizamos o total de $N_{te} = 128$ experimentos.

As seções seguintes discutem os principais resultados das soluções propostas nesses experimentos.

5.3.2 Resultados dos Escalonadores

Os escalonadores propostos (LWS, LTS e HSR) foram avaliados em comparação com o escalonador padrão (LL) do MPTCP em cada um dos quatro algoritmos de controle de congestionamento (LIA, OLIA, BALIA e wVegas). O indicador de desempenho analisado foi também o de vazão multi-fluxo. Para uma comparação geral dos escalonadores propostos, geramos gráficos de barra e *boxplot* nas Figuras 16 e 17, comparando os escalonadores pela vazão geral obtida dos experimentos, em cada controle de congestionamento.

O gráfico da Figura 16(a) mostra o resultado geral dos experimentos usando o controle de congestionamento LIA. No geral, houve um leve ganho do HSR sobre o LL, com resultados mais detalhados no *boxplot* da Figura 18. Em LIA, o fator de agressividade limita o crescimento da janela dos sub-fluxos, sendo um fator único para todos os sub-fluxos estabelecidos na conexão MPTCP. Em tal fator de agressividade há a condição definida para que a janela de um sub-fluxo não cresça mais que a janela do melhor sub-fluxo. Mesmo os escalonadores priorizando os melhores sub-fluxos, o fator de agressividade pode ter seu valor calculado de modo a



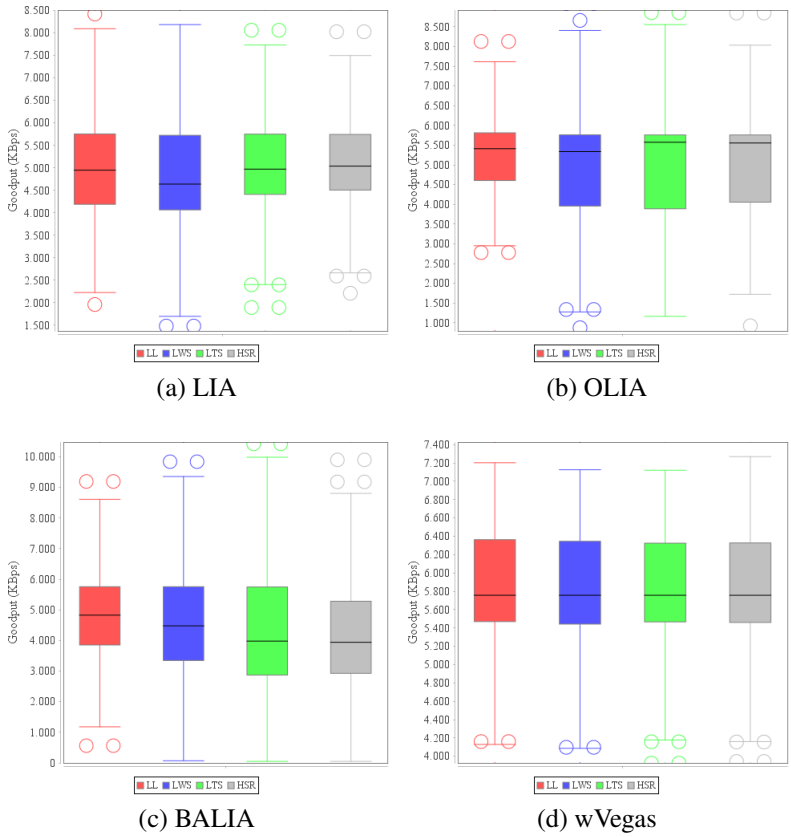
(a) LIA (b) OLIA

(c) BALIA (d) wVegas

Figura 16 – Média da vazão multi-fluxo obtida de todos experimentos com os controle de congestionamento LIA, OLIA, BALIA e wVegas sobre o gerenciador de caminhos *Full-mesh* utilizando todos os escalonadores.

limitar a vazão mais do que deveria. Essa hipótese pode ser verificada através dos Experimentos 4 e 2, em que os resultados HSR são similares nos dois experimentos. Vale destacar que, nesse caso, no Experimento 4 a configuração da taxa de transmissão é heterogênea, sendo a soma das capacidades de transmissão muito superior ao da configuração homogênea no Experimento 2.

Para o controle de congestionamento OLIA, o escalonador HSR obteve maior vazão média, conforme mostram os gráficos das Figura 16 e 17. Uma razão para tal é que no Experimento 4, HSR obteve desempenho consideravelmente melhor que os demais, conforme mostra o gráfico da Figura 19. O HSR utiliza como base para comparação a taxa de transmissão estimada dos sub-fluxos, permitindo melhor diferenciar e selecionar os sub-fluxos de capacidades heterogêneas. O HSR, LWS e LTS foram melhores que o LL, pois o latência heterogênea é crescente respectivamente com a taxa de transmissão heterogênea. Assim, quando o LL deveria escalonar o sub-fluxo com maior taxa de transmissão, este seleciona o de menor latência, que, no caso da parametrização dos enlaces, é o sub-fluxo que percorre o caminho de menor taxa de transmissão. No Experimento 3, o LWS sobressai sobre todos os escalonadores comparados,



(a) LIA (b) OLIA

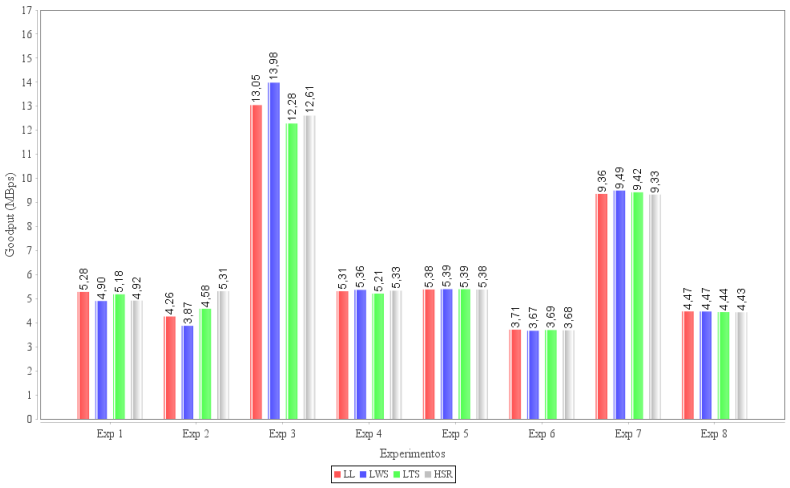
(c) BALIA (d) wVegas

Figura 17 – *Boxplot* da vazão multi-fluxo obtida de todos experimentos com os controle de congestionamento LIA, OLIA, BALIA e wVegas sobre o gerenciador de caminhos *Full-mesh* utilizando todos os escalonadores.

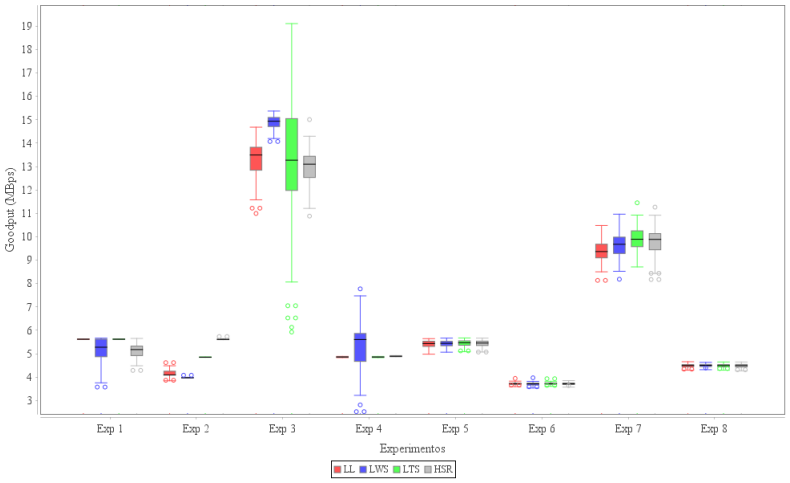
pois sua estratégia é a de escolher o sub-fluxo de menor espaço na janela, onde os sub-fluxos mais rápidos tendem a ter mais espaço na janela.

Nos experimentos realizados sobre o controle de congestionamento BALIA, o escalonador LL obteve maior vazão de média, como mostra a Figura 16. De modo mais detalhado, o gráfico da Figura 20 mostra os resultados de cada experimento. O LL obteve tal resultado, pois se favoreceu da estratégia de controle de congestionamento em BALIA. Quando BALIA reduz o tamanho da janela mais agressivamente que o padrão do TCP, prejudica os escalonadores propostos, pois estes, de alguma forma estão utilizando a janela de congestionamento como base de comparação dos sub-fluxos. O LL, entretanto, utiliza isoladamente a latência prevista via sRTT dos sub-fluxos como base de comparação. No Experimento 3, observamos que o LWS obteve maior vazão. Como nesse experimento não há perda de pacote nos enlaces mas há taxa de transmissão muito heterogênea e relativamente grande entre os enlaces, a redução de janela em BALIA não exerceu degradação significativa na vazão, como observado nos outros escalonadores. No Experimento 4, observamos o LTS que a estratégia de escolher o sub-fluxo com base na razão entre o espaço na janela e latência possibilitou maior desempenho com BALIA.

Por fim, no controle de congestionamento wVegas, observamos que os resultados são



(a)

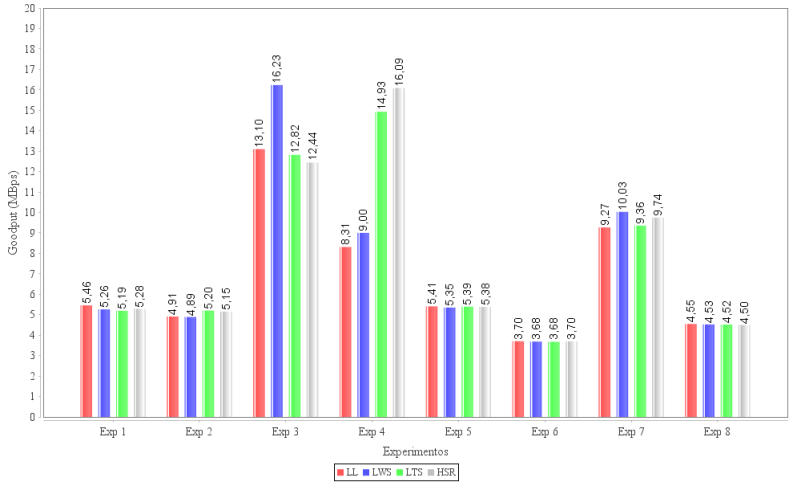


(b)

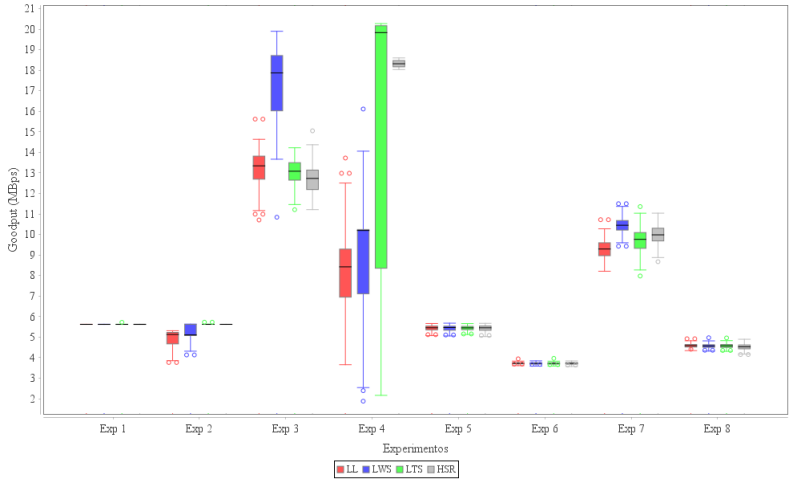
Figura 18 – Comparação entre LL, LWS, LTS e HSR, utilizando o controle de congestionamento LIA e gerenciador *Full-mesh*.

muito similares entre os escalonadores, como apresenta a Figura 16. Isso é devido à estratégia de controle de congestionamento em wVegas, que não depende unicamente do evento de perda e utiliza majoritariamente a estimativa do tempo de enfileiramento dos caminhos para ajustar a janela. Tal estratégia possibilita transferir as cargas dos sub-fluxos mais congestionados para os menos congestionados, independentemente da decisão de escalonamento. Porém, ainda podemos analisar sutis diferenças entre os escalonadores, conforme mostra a Figura 21. No Experimento 3, observamos o HSR e o LWS com maior vazão devido a parametrização dos enlaces, conforme discutimos anteriormente. Já no experimento 4, observamos que a estratégia LTS, de escolher sub-fluxos com base na menor relação entre o maior espaço da janela e a menor estimativa de tempo de ida e volta, possibilitou maior vazão. No Experimento 7, mesmo em situações com perda de pacote, o escalonador LWS selecionou sub-fluxos mais aptos possibilitando também desempenho médio levemente superior aos demais escalonadores.

Analisando de forma geral os resultados obtidos, é possível ver que nos Experimentos 5,



(a)



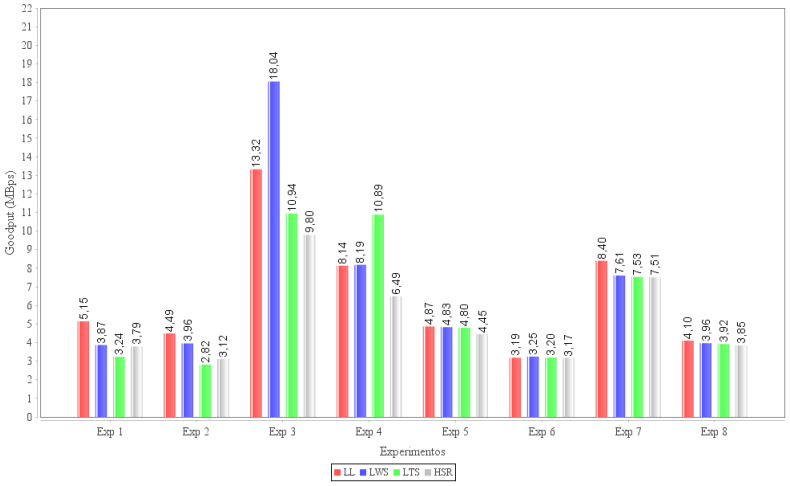
(b)

Figura 19 – Comparação entre LL, LWS, LTS e HSR, utilizando o controle de congestionamento OLIA e gerenciador *Full-mesh*.

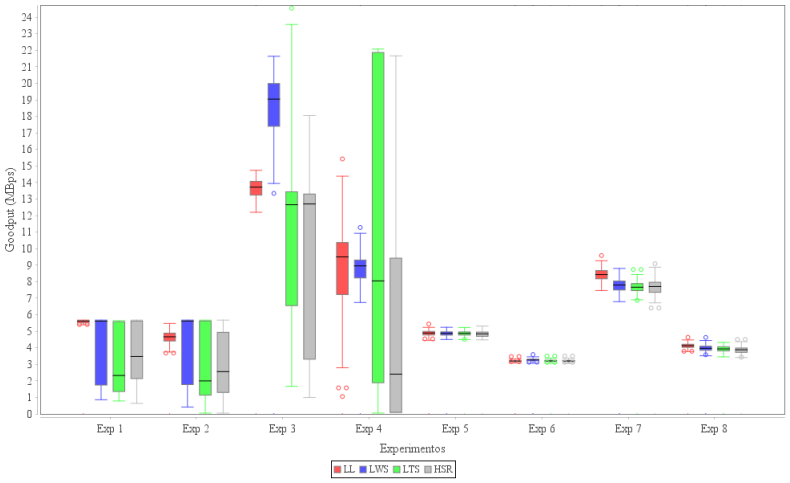
6, 7 e 8, a vazão dos escalonadores são similares, indentificando uma influência comum gerada principalmente pela perda de pacote parametrizada de forma heterogênea. Nos Experimentos 6 e 8, a composição de um cenário misto, com perda de pacote e maior latência nos caminhos, degrada consideravelmente a eficiência da escolha dos sub-fluxos pelos escalonadores, mesmo sobre caminhos de capacidades variadas de transmissão, como no Experimento 8. No Experimento 7, mesmo tendo a parametrização heterogênea mista entre perda de pacote e latência, os resultados são próximos, com diferenças sutis na vazão resultante dos escalonadores. Pretendemos atacar esses cenários em investigações futuras.

5.3.3 Resultados do Gerenciadores de Caminhos

Seguindo a mesma metodologia dos experimentos realizados para avaliação de desempenho dos escalonadores sobre caminhos de características homogêneas e heterogêneas, o gerenciador de caminho proposto, *Single-mesh*, foi avaliado e comparado com gerenciador padrão



(a)

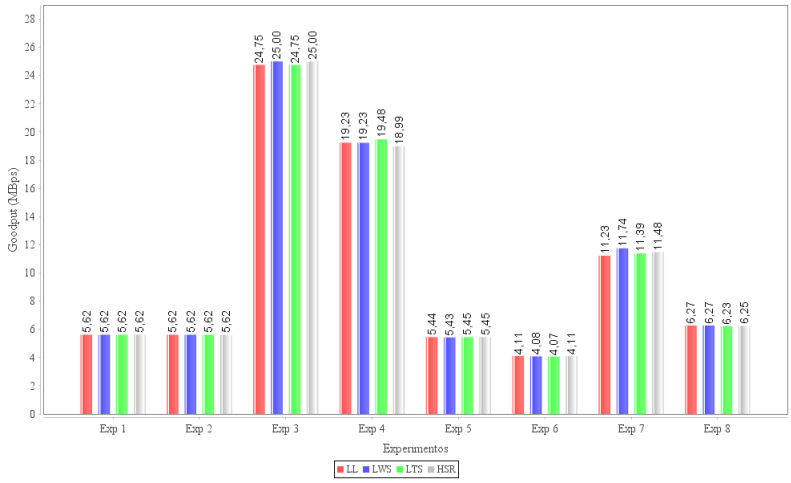


(b)

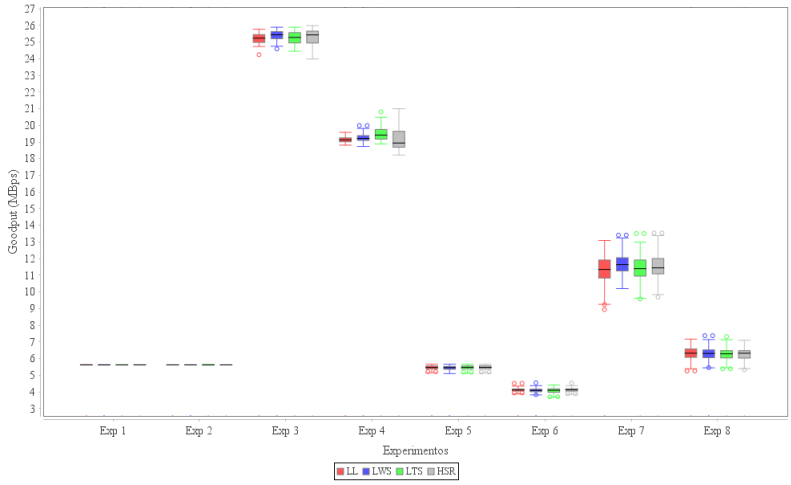
Figura 20 – Comparação entre LL, LWS, LTS e HSR, utilizando o controle de congestionamento BALIA e gerenciador *Full-mesh*.

do MPTCP, *Full-mesh*. Nesta seção, discutimos o resultados de *Single-mesh* obtidos com o uso do escalonador LL, em cada dos quatro algoritmos de controle de congestionamento (LIA, OLIA, BALIA e wVegas). O indicador de desempenho analisado também foi de vazão multi-fluxo. Os resultados obtidos são apresentados nos gráficos das Figuras 22, 23, 24 e 25.

Conforme os experimentos realizados, verificamos que *Single-mesh* obteve pior desempenho em relação ao *Full-mesh* em todas as configurações dos experimentos utilizando os controles de congestionamento LIA e wVegas, como são apresentados nos gráficos da Figura 22 e 25. No caso de LIA, isso ocorreu devido ao fato de o mesmo ser mais agressivo no crescimento da janela de congestionamento em relação aos outros algoritmos de controle de congestionamento. Embora a percepção de congestionamento entre os sub-fluxos no caminho reduza agressivamente as janelas dos sub-fluxos em $w_r/2$, a expansão das janelas força maior utilização dos caminhos. O wVegas não beneficia o uso do *Single-mesh*, pois seu algoritmo regula a janela w_r dos sub-fluxos com maior sensibilidade às condições da rede através atraso de en-



(a)

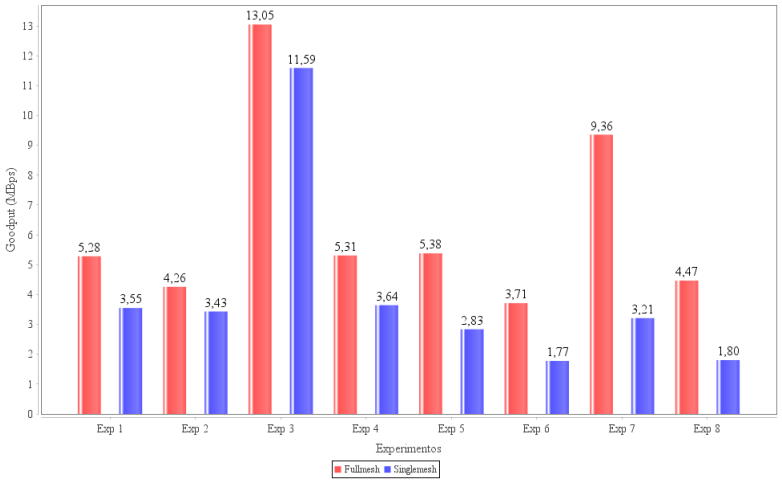


(b)

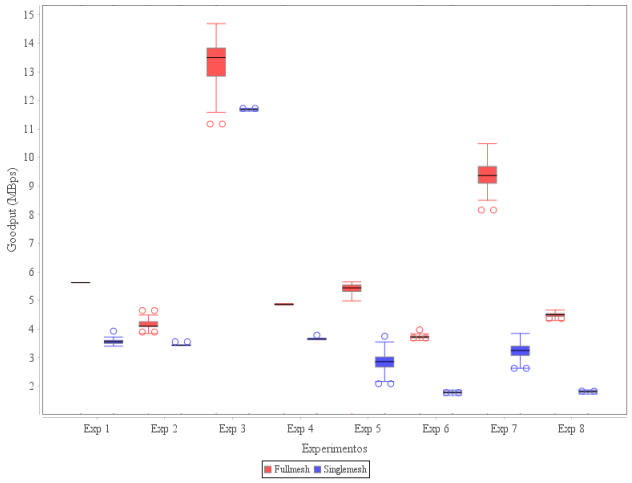
Figura 21 – Comparação entre LL, LWS, LTS e HSR, utilizando o controle de congestionamento wVegas e gerenciador *Full-mesh.*

fileiramento dos enlaces. Isso possibilita maior responsividade aos sub-fluxos, ao passo que o congestionamento causado pelos sub-fluxos num caminho é drenado e redirecionando para outros sub-fluxos que percorrem caminhos menos ocupados.

Por outro lado, para os controles de congestionamento OLIA e BALIA, apresentados nos gráficos da Figura 23 e 24, respectivamente, o *Single-mesh* apresentou ganho de desempenho em cenários nos experimentos 1, 2, 3 e 4. Nos experimentos 3 e 4 o ganho de Single-mesh foi expressivo, de pelo menos o dobro da vazão de *Full-mesh*. Nesses experimentos, a taxa de perda dos caminhos é homogênea em 0 %, de modo que o congestionamento nos caminhos é gerado pela expansão das janelas dos próprios sub-fluxos. No caso de OLIA e BALIA, a expansão das janelas é menos agressiva. Ao inferir congestionamento através da perda de pacote no caminho, a redução das janelas de congestionamento dos sub-fluxos causa maior impacto no desempenho, principalmente em BALIA. Como observado, o congestionamento gerado entre os 5 sub-fluxos por caminho no *Full-mesh* é tal que o desempenho é inferior ao de um único sub-



(a)

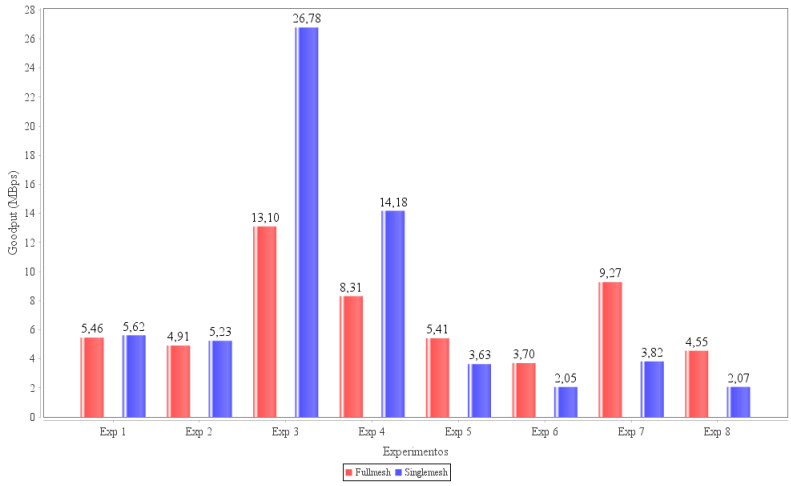


(b)

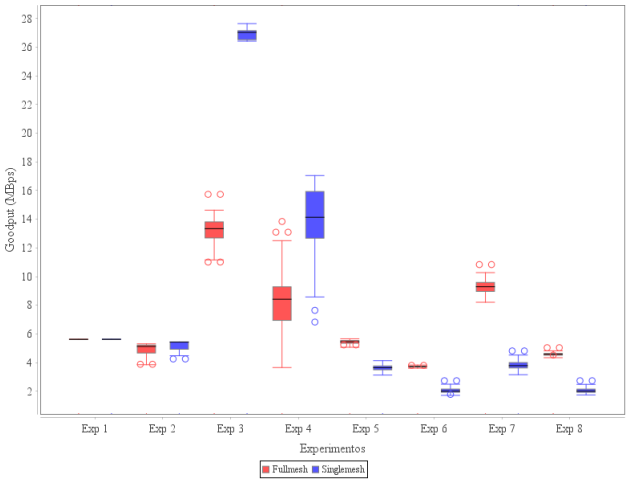
Figura 22 – Comparação do *Single-mesh* e *Full-mesh*, utilizando o controle de congestionamento LIA.

fluxo por caminho em *Single-mesh*. Ao limitar o estabelecimento de no máximo um sub-fluxo por caminho entre cliente e servidor MPTCP, evita-se o congestionamento entre os sub-fluxos, possibilitando que o sub-fluxo estabelecido concorra apenas com outros fluxos não-MPTCP que compartilham o mesmo caminho. Como observado, *Single-mesh* obteve melhor desempenho agregado de todo multi-fluxo em cenários de baixa perda de pacotes. Ainda, os resultados de *Single-mesh* nos experimentos 1, 2, 3 e 4 com os controles de congestionamento OLIA e BALIA mostraram-se próximos ou superiores aos de *Full-mesh* em wVegas, o qual foi o mais eficiente algoritmo de controle de congestionamento, de acordo com os outros experimentos realizados.

Para cenários de taxa de perda heterogênea nos experimentos 5, 6, 7 e 8, *Single-mesh* foi pior em relação ao *Full-mesh* em todos os algoritmos de controle de congestionamento. Isso é explicado pelo fato que os sub-fluxos adicionais de *Full-mesh* possibilitam a ele uma nova tentativa de envio dos pacotes, ou seja, o escalonador pode escolher outros sub-fluxos que não



(a)

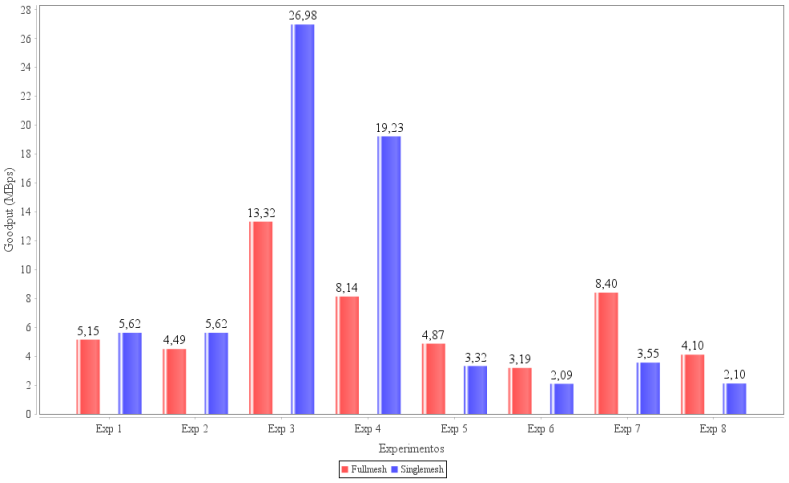


(b)

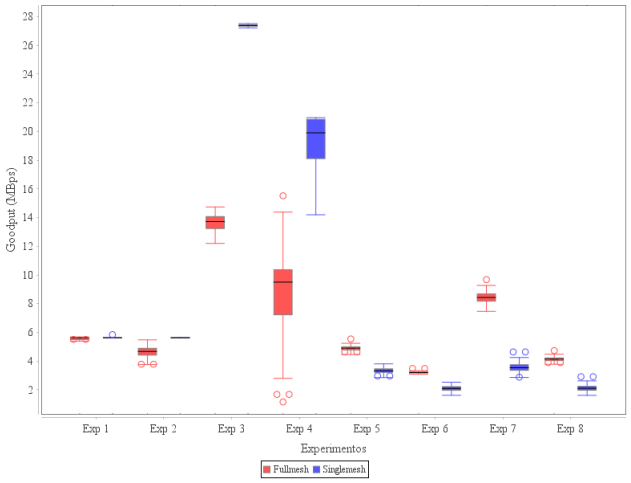
Figura 23 – Comparação do *Single-mesh* e *Full-mesh*, utilizando o controle de congestionamento OLIA.

estejam congestionados (como por exemplo, que não estão sob estado de perda) no instante do envio.

Vale destacar que *Single-mesh* apresenta eficiência em situações onde não existe concorrência de outras aplicações pelo uso dos caminhos da rede. Em um ambiente real, onde inúmeras aplicações utilizam caminhos compartilhados na rede, o uso do *Full-mesh* pode gerar gargalos mais rapidamente, afetando outras aplicações que não utilizam o MPTCP. Nesse caso, pelo fato de diminuir a quantidade de sub-fluxos e não estabelecer mais de um sub-fluxo por caminho, *Single-mesh* não prejudica a transmissão de outras aplicações, sendo mais amistoso no compartilhamento de enlaces de gargalo com outros fluxos TCP de caminhos únicos.



(a)

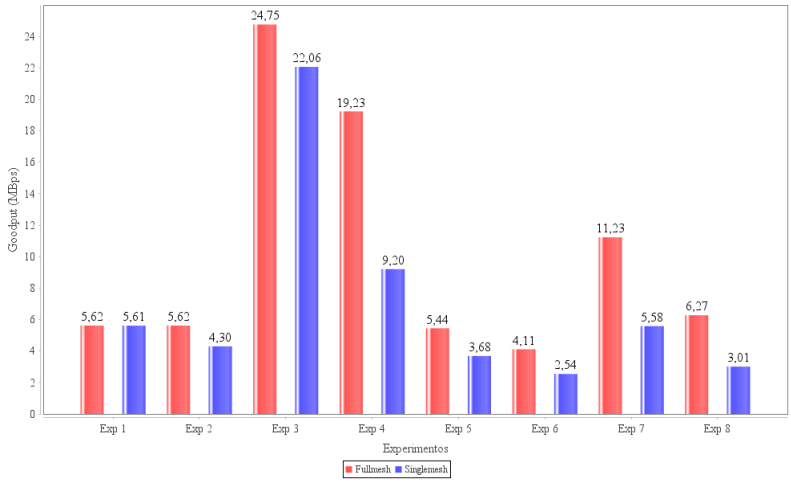


(b)

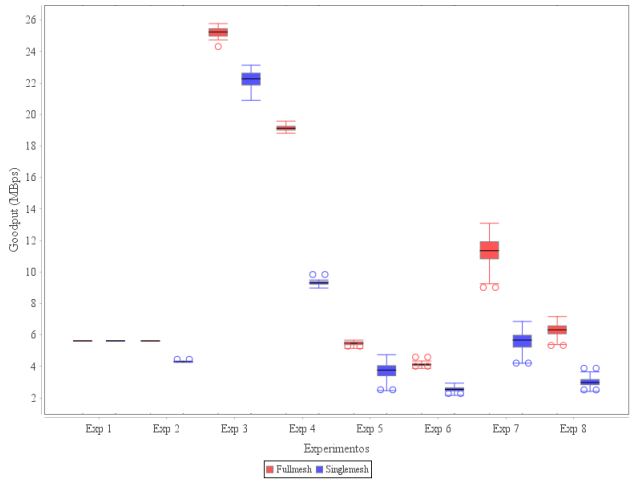
Figura 24 – Comparação do *Single-mesh* e *Full-mesh*, utilizando o controle de congestionamento BALIA.

5.4 Considerações Finais

Neste capítulo, descrevemos o ambiente experimental configurado, as ferramentas utilizadas, como configuramos e parametrizamos os experimentos, e as metodologias utilizadas no planejamento dos experimentos. Para cada metodologia, discutimos os resultados obtidos para cada solução proposta, dos escalonadores alternativos de pacotes ao novo gerenciador de caminhos. Os resultados obtidos validam as soluções propostas, uma vez que elas melhoram o desempenho do MPTCP, mitigando o impacto dos problemas identificados. Observamos, contudo, que cada solução mostra-se mais eficiente para um determinado tipo de cenário, como também o desempenho de cada solução depende do controle de congestionamento escolhido. A exceção identificada foi no controle de congestionamento wVegas, em que as soluções de escalonamento não exercem muita influência no desempenho final, provendo *goodput* nivelado, devido à estratégia abordada de ajuste da janela de controle de congestionamento dos sub-fluxos com base na estimativa de latência de enfileiramento dos caminhos.



(a)



(b)

Figura 25 – Comparação do *Single-mesh* e *Full-mesh*, utilizando o controle de congestionamento wVegas.

Conforme os resultados obtidos, observamos que o escalonador LWS é a melhor escolha para cenários onde os caminhos possuem capacidade de transmissão distinta. O escalonador LTS, por outro lado, se mostrou mais apto para cenários com latência distinta entre os caminhos sob os controles de congestionamento OLIA e BALIA. O escalonador HSR, por sua vez, foi melhor apenas com o uso do controle de congestionamento OLIA e para cenários com latência heterogênea entre os caminhos usados pelos sub-fluxos. Por fim, o novo gerenciador de caminhos proposto, *Single-mesh*, é uma boa escolha que possbilidade maior amistosidade no uso de recursos compartilhados, ao passo que melhora o desempenho das transmissão em multi-fluxos sob os controles de congestionamento OLIA e BALIA em situações que não haja perda de pacotes nos caminhos. Nesse contexto, o conteúdo deste capítulo, fornece ao leitor uma análise detalhada dos resultados obtidos para cada solução proposta, permitindo ao leitor compreender o uso das soluções para cada tipo de cenário.

6 CONCLUSÕES

O transporte de dados fim-a-fim através de múltiplos caminhos em MPTCP permite dois benefícios imediatos: maior desempenho e maior tolerância a falhas de transmissão. Embora tais benefícios permitam melhorar consideravelmente as comunicações na Internet, o protocolo MPTCP ainda demanda aprimoramentos em seus mecanismos internos, principalmente, na eficiência do emissor.

Neste trabalho, identificamos experimentalmente problemas de desempenho dos mecanismos de escalonamento de sub-fluxos e do gerenciamento de caminhos do MPTCP. Os problemas observados estão relacionados às condições dos caminhos e as respectivas tomadas de decisão desses mecanismos. O escalonador padrão, LL, ao priorizar apenas sub-fluxos de menor estimativa de latência, pode não proporcionar maior ganho de transmissão agregada em situações onde os caminhos existentes possuem características distintas, em termos de capacidade de transmissão, taxa de perda de pacote e a latência fim-a-fim. O gerenciador de caminhos *Full-mesh*, ao estabelecer uma malha completa de sub-fluxos sobre os caminhos existentes entre os sistemas finais envolvidos, gera concorrência entre os próprios sub-fluxos que compartilham um mesmo caminho. Em caminhos não congestionados, livres de perda de pacotes, tal concorrência gera degradação de desempenho multi-fluxo.

Para contornar esses problemas, então propomos, implementamos e avaliamos um conjunto de soluções alternativas em um ambiente experimental. Tal ambiente foi implantado a partir de uma topologia ponto-a-ponto, onde dois sistemas finais MPTCP foram ligados através de cinco interfaces de rede em cada ponto, possibilitando 5 caminhos físicos e logicamente distintos entre eles. Embora minimalista, esse ambiente foi satisfatório para a investigação que realizamos, uma vez que foi possível selecionar fatores e domínio de valores para parametrizar os caminhos, gerando diversas condições de rede que o MPTCP enfrentaria um ambiente real na Internet.

A partir de uma quantidade considerável de experimentos que caracterizaram situações diversas de caminhos fim-a-fim, observamos que os escalonadores propostos, LWS, LTS e HSR, obtiveram resultados promissores. Tais resultados evidenciam que novas alternativas de escalonamento de sub-fluxos podem ser utilizadas pelo MPTCP com melhor desempenho e aproveitamento dos recursos de rede que o escalonador padrão LL. Nesses resultados, observamos ainda que o controle de congestionamento, ao regular a expansão e retração das janelas dos sub-fluxos de acordo com objetivos particulares dos algoritmos, são capazes de mover fluxo de dados de sub-fluxos mais congestionados para os menos congestionados. Essa operação impacta diretamente no desempenho das transmissões em multi-fluxos e, algumas vezes, diminuem a influência que a decisão de escalonamento tem sobre a vazão final obtida, como foi observado nos

experimentos com wVegas.

Os resultados obtidos com o gerenciador de caminhos proposto, *Single-mesh*, foram igualmente interessantes. Uma vez que apenas um sub-fluxo MPTCP é estabelecido por caminho, o impacto negativo do congestionamento entre os sub-fluxos observado em *Full-mesh* não ocorre em *Single-mesh*. Além de ser mais justo e amistoso no compartilhamento de recursos de rede, *Single-mesh* obteve melhor desempenho agregado em transmissão multi-fluxo sobre caminhos de baixa perda de pacotes, tipicamente caminhos não congestionados.

O MPTCP é um protocolo ainda em desenvolvimento, atraindo atenção da academia e da indústria na busca de melhorias, tanto no desempenho quanto nas aplicações do protocolo para resolver problemas diversos de comunicação de dados, por exemplo, mobilidade e tolerância a falha. Neste trabalho pudemos explorar no campo da pesquisa as questões relacionadas ao desempenho, investigando problemas, propondo e avaliando soluções em ambiente experimental.

6.1 Contribuições

Entendemos que a principal contribuição deste trabalho é a discussão, que vai desde a identificação de problemas de degradação de desempenho de transmissões em multi-fluxos MPTCP até a concepção e validação de soluções alternativas mais eficientes para o protocolo. A partir de algoritmos simples que não alteram nenhuma das propriedades dos algoritmos existentes de controle de congestionamento, as soluções propostas possibilitam maior desempenho que as soluções padrão do MPCTP, permitindo maior vazão fim-a-fim em diversos cenários de comunicação de multi-fluxo na Internet.

A expectativa é que essa discussão possa ser útil tanto para a comunidade acadêmica quanto para a indústria. No contexto acadêmico, os principais resultados obtidos com escalonadores propostos foram discutidos no artigo intitulado "*Alternative Scheduling Decisions for Multipath TCP*", o qual foi publicado[1] em Agosto de 2017 na revista *IEEE Communications Letters*, que é uma das principais revistas internacionais existentes atualmente na área de Redes de Computadores. Na contexto da indústria, esperamos que, em algum momento, as soluções propostas venham a ser incorporadas na implementação oficial do MPTCP[2].

Nesse contexto, além do aprimoramento do protocolo MPTCP, transmissões de multi-fluxos mais eficientes proporcionadas pelas soluções alternativas que criamos poderão beneficiar:

- pesquisadores que investigam meios de tornar comunicações mais eficientes em Redes de Computadores;

[1] <https://doi.org/10.1109/LCOMM.2017.2740918>
[2] *Linux Kernel MultiPath TCP Project*: <http://multipath-tcp.org/>

- todo usuário em potencial, que possua dispositivo fixo ou móvel com mais de uma interface de rede para acessar a Internet;
- *datacenters*, onde houver, por exemplo, aplicações de *clusters* e *grids* que demandam maior taxa de transmissão de dados.

6.2 Trabalhos futuros

Nos resultados analisados, observamos que desempenho dos escalonadores propostos LWS, LTS e HSR, dependem das condições atuais dos caminhos fim-a-fim e apresentam maior eficiência em cenários específicos. Como trabalho futuro, pretendemos investigar estratégias para definição de um escalonador dinâmico, de mais alto nível, o qual seja capaz de: $i)$ identificar as condições atuais dos caminhos, sendo sensível às mudanças de características de taxa de transmissão, perda de pacote e latência; $ii)$ e, então, alternar a operação entre os escalonadores propostos, selecionando aquele que provê melhor desempenho, dada a condição atual de cada caminho.

Outro ponto que merece investigação futura é na avaliação de desempenho, tanto dos escalonadores propostos quanto de *Single-mesh*. Espera-se avaliar as soluções propostas em cenários de comunicação onde os caminhos são compartilhados também por fluxos não originados por conexões MPTCP, como fluxos TCP e UDP.

REFERÊNCIAS

ALLMAN, M.; PAXSON, V. On estimating end-to-end network path properties. In: *ACM SIGCOMM Computer Communication Review*. [S.l.: s.n.], 1999. v. 29, n. 4, p. 263–274. Citado na página 12.

ALLMAN, M.; PAXSON, V.; BLANTON, E. Tcp congestion control. *IETF RFC 5681*, 2009. Citado 2 vezes nas páginas 13 e 25.

ARZANI, B. et al. Impact of path characteristics and scheduling policies on mptcp performance. In: *28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2014)*. [S.l.: s.n.], 2014. p. 743–748. Citado 2 vezes nas páginas 41 e 42.

BISWAS, S. et al. Large-scale Measurements of Wireless Network Behavior. In: *ACM SIGCOMM Computer Communication Review*. [S.l.: s.n.], 2015. v. 45, n. 4, p. 153–165. Citado na página 63.

BOCCASSI, L.; FAYED, M. M.; MARINA, M. K. Binder: A system to aggregate multiple internet gateways in community networks. In: *Proceedings of the 2013 ACM MobiCom workshop on Lowest cost denominator networking for universal access*. [S.l.: s.n.], 2013. p. 3–8. Citado na página 35.

BONAVENTURE, O.; PAASCH, C.; DETAL, G. Use Cases and Operational Experience with Multipath TCP. *IETF RFC 8041*, 2017. Citado 4 vezes nas páginas 1, 2, 20 e 36.

BONAVENTURE, O.; SEO, S. Multipath tcp deployments. *IETF Journal*, v. 12, n. 2, p. 24–27, 2016. Citado na página 2.

BOVY, C. et al. Analysis of end-to-end delay measurements in internet. In: *Proc. of the Passive and Active Measurement Workshop-PAM*. [S.l.: s.n.], 2002. v. 2002. Citado na página 63.

BROWN, M. A. Traffic control howto. *Guide to IP Layer Network*, 2006. Citado na página 49.

CAO, Y.; XU, M.; FU, X. Delay-based congestion control for multipath tcp. In: *IEEE International Conference on Network Protocols (ICNP)*. [S.l.: s.n.], 2016. p. 1–10. Citado 3 vezes nas páginas 6, 29 e 31.

CHEN, Y.-C. et al. A Measurement-based Study of Multipath TCP Performance over Wireless Networks. In: *ACM Conference on Internet Measurement Conference*. [S.l.: s.n.], 2013. p. 455–468. Citado na página 63.

FORD, A. et al. Architectural guidelines for multipath tcp development. *IETF RFC 6182*, 2011. Citado 8 vezes nas páginas 1, 2, 14, 18, 21, 31, 34 e 40.

FORD, A. et al. Tcp extensions for multipath operation with multiple addresses. *IETF RFC 6824*, 2013. Citado 5 vezes nas páginas 11, 1, 9, 14 e 15.

FOUNDATION, C. L. U. Ubuntu (operating system). 2010. Citado na página 48.

FUENTES, F.; KAR, D. C. Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose. *Journal of Computing Sciences in Colleges*, v. 20, n. 4, p. 169–176, 2005. Citado na página 50.

HERMAN, P. The tcpstat tool. *Publicly available at: http://www. frenchfries. net/paul/tcpstat*, 2001. Citado na página 52.

HUNGER, A.; KLEIN, P. Equalizing latency peaks using a redundant multipath-tcp scheme. In: *2016 IEEE International Conference on Information Networking (ICOIN 2016)*. [S.l.: s.n.], 2016. p. 184–189. Citado na página 33.

HWANG, J.; YOO, J. Packet scheduling for multipath tcp. In: *2015 7th IEEE International Conference on Ubiquitous and Future Networks*. [S.l.: s.n.], 2015. p. 177–179. Citado na página 43.

IYENGAR, J. R.; AMER, P. D.; STEWART, R. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Transactions on networking*, v. 14, n. 5, p. 951–964, 2006. Citado na página 9.

JACOBSON, V. Congestion avoidance and control. *ACM SIGCOMM Computer Communication Review*, v. 25, n. 1, p. 157–187, 1995. Citado 3 vezes nas páginas 12, 32 e 33.

JURGELIONIS, A. et al. An empirical study of netem network emulation functionalities. In: *Proceedings of 20th IEEE International Conference on Computer Communications and Networks (ICCCN 2011)*. [S.l.: s.n.], 2011. p. 1–6. Citado na página 50.

KARN, P.; PARTRIDGE, C. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM SIGCOMM Computer Communication Review*, ACM, v. 17, n. 5, p. 2–7, 1987. Citado na página 32.

KE, F. et al. Multi-attribute aware multipath data scheduling strategy for efficient mptcp-based data delivery. In: *22nd IEEE Asia-Pacific Conference on Communications (APCC 2016)*. [S.l.: s.n.], 2016. p. 248–253. Citado na página 43.

KHALILI, R. et al. Opportunistic linked-increases congestion control algorithm for mptcp. *IETF, Individual Submission, Internet Draft draft-khalili-mptcp-congestion-control-02*, 2013. Citado 2 vezes nas páginas 6 e 27.

LAMPING, U.; SHARPE, R.; WARNICKE, E. *Wireshark Users Guide for Wireshark 2.1*. 2014. Citado na página 51.

LE, T.-A.; BUI, L. X. Forward delay-based packet scheduling algorithm for multipath tcp. *arXiv preprint arXiv:1501.03196*, 2015. Citado na página 45.

LIU, C. L.; LAYLAND, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, ACM, v. 20, n. 1, p. 46–61, 1973. Citado na página 2.

LOPEZ, I. et al. Scada systems in the railway domain: enhancing reliability through redundant multipathtcp. In: *18th IEEE International Conference on Intelligent Transportation Systems (ITSC)*. [S.l.: s.n.], 2015. p. 2305–2310. Citado na página 34.

MEHANI, O. et al. An early look at multipath tcp deployment in the wild. In: *Proceedings of the 6th ACM International Workshop on Hot Topics in Planet-Scale Measurement*. [S.l.: s.n.], 2015. p. 7–12. Citado na página 36.

NGUYEN, H. X.; ROUGHAN, M. Rigorous statistical analysis of internet loss measurements. *IEEE/ACM Transactions on Networking (TON)*, v. 21, n. 3, p. 734–745, 2013. Citado na página 63.

NI, D. et al. Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath tcp in lossy networks. In: *23rd International Conference on Computer Communication and Networks (ICCCN 2014)*. [S.l.: s.n.], 2014. p. 1–7. Citado na página 44.

NIKRAVESH, A. et al. An In-depth Understanding of Multipath TCP on Mobile Devices: Measurement and System Design. In: *22nd Annual ACM International Conference on Mobile Computing and Networking*. [S.l.: s.n.], 2016. p. 189–201. Citado na página 63.

OH, B.-H.; LEE, J. Constraint-based proactive scheduling for mptcp in wireless networks. *Computer Networks*, Elsevier, v. 91, p. 548–563, 2015. Citado 2 vezes nas páginas 44 e 45.

PAASCH, C.; BONAVENTURE, O. et al. Decoupled from ip, tcp is at last able to support multihomed hosts. *ACM Queue (Print): tomorrow's computing today*, v. 12, 2014. Citado 2 vezes nas páginas 11 e 16.

PAASCH, C. et al. The fastest TCP connection with MultiPath TCP. In: . [s.n.], 2014. Disponível em: <http://multipath-tcp.org/pmwiki.php?n=Main.50Gbps>. Citado 2 vezes nas páginas 5 e 63.

PAASCH, C. et al. Experimental Evaluation of Multipath TCP Schedulers. In: *ACM SIGCOMM Workshop on Capacity Sharing Workshop*. [S.l.: s.n.], 2014. p. 27–32. Citado 5 vezes nas páginas 5, 31, 32, 33 e 42.

PAASCH, C.; KHALILI, R.; BONAVENTURE, O. On the Benefits of Applying Experimental Design to Improve Multipath TCP. In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. [S.l.: s.n.], 2013. (CoNEXT '13), p. 393–398. Citado 2 vezes nas páginas 5 e 63.

PAXSON, V.; ALLMAN, M. Computing tcp's retransmission timer. *IETF RFC 2988*, 2000. Citado na página 12.

PENG, Q. et al. Multipath tcp: Analysis, design, and implementation. *IEEE/ACM Transactions on Networking (ToN)*, v. 24, n. 1, p. 596–609, 2016. Citado na página 19.

POSTEL, J. et al. Transmission control protocol. *IETF RFC 793*, 1981. Citado 4 vezes nas páginas 11, 9, 10 e 12.

RAICIU, C. et al. Improving datacenter performance and robustness with multipath tcp. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2011. v. 41, n. 4, p. 266–277. Citado na página 1.

RAICIU, C.; HANDLEY, M.; WISCHIK, D. Coupled congestion control for multipath transport protocols. *IETF RFC 6356*, 2011. Citado 3 vezes nas páginas 6, 19 e 25.

SINGH, A. et al. Performance comparison of scheduling algorithms for multipath transfer. In: *IEEE Global Communications Conference (GLOBECOM 2012)*. [S.l.: s.n.], 2012. p. 2653–2658. Citado na página 42.

TSAI, M.-H.; CHOU, C.-M.; LAN, K.-c. Avoiding biased-feeding in the scheduling of collaborative multipath tcp. *PloS one*, v. 11, n. 8, p. e0161213, 2016. Citado na página 45.

WALID, A. et al. Balanced linked adaptation congestion control algorithm for mptcp. *IETF, Individual Submission, Internet Draft draft-walid-mptcp-congestion-control-03*, 2015. Citado 2 vezes nas páginas 6 e 28.

WANG, K. et al. On the path management of multi-path tcp in internet scenarios based on the nornet testbed. In: IEEE. *Advanced Information Networking and Applications (AINA), 2017 IEEE 31st International Conference on*. [S.l.], 2017. p. 1–8. Citado na página 35.

YANG, F.; AMER, P.; EKIZ, N. A scheduler for multipath tcp. In: *22nd International Conference on Computer Communications and Networks (ICCCN 2013)*. [S.l.: s.n.], 2013. p. 1–7. Citado na página 42.

YANG, F.; WANG, Q.; AMER, P. D. Out-of-order transmission for in-order arrival scheduling for multipath tcp. In: *28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2014)*. [S.l.: s.n.], 2014. p. 749–752. Citado na página 43.

YEDUGUNDLA, K. et al. Is multi-path transport suitable for latency sensitive traffic? *Computer Networks*, Elsevier, v. 105, p. 1–21, 2016. Citado 2 vezes nas páginas 21 e 34.

ZHANG, B. et al. Measurement-Based Analysis, Modeling, and Synthesis of the Internet Delay Space. *IEEE/ACM Transactions on Networking*, v. 18, n. 1, p. 229–242, Feb 2010. Citado na página 63.

Apêndices

APÊNDICE A – CONSIDERAÇÕES PRÁTICAS DAS SOLUÇÕES PROPOSTAS

A.1 Escalonador LWS

LWS foi implementado no sub-módulo de escalonamento do MPTCP para substituir a operação do escalonador padrão, LL, utilizando, contudo, as mesmas verificações de disponibilidade dos sub-fluxos. O Código A.1 apresenta a sua implementação no laço principal do sub-módulo de escalonamento do MPTCP.

A variável `tp` (linha 2), declarada com tipo definido pela estrutura `tcp_sock`, possui as dados correntes sobre o sub-fluxo r, nesse caso, indicado pela variável `sk`. O cálculo do espaço na janela é indicado na linha 34, onde `snd_cwnd` é o tamanho da janela de congestionamento atual e `tcp_packets_in_flight()` é uma função que retorna o número de pacotes que estão sob transmissão, ainda pendentes de reconhecimento `ACK`. O trecho de código da linha 5 a 15 representa a decisão de escalonamento LWS, conforme sua descrição no Algoritmo 4.

Código A.1 – Implementação de LWS no sub-módulo de escalonamento do MPTCP.

```
1  mptcp_for_each_sk(mpcb, sk) {
2     struct tcp_sock *tp = tcp_sk(sk);
3     // Inicializações e verificações
4     // Cálculo do espaço na janela
5     space = tp->snd_cwnd - tcp_packets_in_flight(tp);
6
7     /* Verifica se o espaço é negativo */
8     if (space <= 0 && tp->srtt_us <= min_srtt) {
9        min_srtt = tp->srtt_us;
10       bestsk = sk;
11    } /* Comparação para achar o melhor path */
12    else if(space >= max_space) {
13       max_space = space;
14       bestsk = sk;
15    }
16    // Fim da implementação
17 }
```

A.2 Escalonador LTS

O trecho de implementação pertinente ao escalonador LTS é apresentado no Código A.2.

Código A.2 – Implementação de LTS no sub-módulo de escalonamento do MPTCP.

```
1  mptcp_for_each_sk(mpcb, sk) {
2     struct tcp_sock *tp = tcp_sk(sk);
3     // Inicializações e verificações
4     // Cálculo do espaço da janela
5     space = tp->snd_cwnd - tcp_packets_in_flight(tp);
6
7     if (space <= 0 && tp->srtt_us <= min_val) {
8        min_val = tp->srtt_us;
9        bestsk = sk;
10    }
11    else{
12    /* Cálculo dos parâmetros sRTT e espaço da janela */
13       val = tp->srtt_us/space;
14
15       if (val <= min_val) {
16          min_val = val;
17          bestsk = sk;
18       }
19    }
20 }
```

A razão τs entre os dois parâmetros é obtida na linha 13 com a variável `val`, onde `srtt_us` é o tempo estimado τ_r de ida e volta e a variável `space` é o espaço da janela ws. Assim como no código de LWS, a variável `tp` é a estrutura do tipo `tcp_sock` que possui os dados de um dado sub-fluxo r. O sub-fluxo escolhido (r_{best}) é aquele que de menor relação tempo/espaço armazenado em `bestsk`.

A.3 Escalonador HSR

A implementação de HSR tem a mesma estrutura que os escalonadores anteriores. A capacidade de transmissão sr do sub-fluxo r é calculada na linha 5, onde `snd_cwnd` é o tamanho atual da janela w_r em número de pacotes, m_r é implementado por `mss_cache`, contendo o tamanho máximo do segmento (MSS) utilizado no sub-fluxo r em *bytes* e, por fim, o `rtt_inst` é o tempo de ida e volta instantâneo $\ddot{\tau}_r$ em microssegundos. A constante 16 é um coeficiente escalar, como o utilizado no controle de congestionamento wVegas, que indica a precisão dos cálculos realizados no espaço de núcleo do SO, pois no Linux não é possível realizar manipular ponto flutuante, impedindo cálculo com números valores reais.

A função `div64()` realiza divisões dois valores de até 64 bits armazenados em variáveis inteiras de 8 bytes. Com as informações atuais do sub-fluxo obtidas da estrutura `tcp_sock` a partir da variável `tp`, a capacidade ws atual do sub-fluxo r é então armazenada na variável `rate`.

Código A.3 – Implementação de HSR no sub-módulo de escalonamento do MPTCP.

```
1  mptcp_for_each_sk(mpcb, sk) {
2     struct tcp_sock *tp = tcp_sk(sk);
3     // Inicializações e verificações
4     // Cálculo da capacidade de transmissão
5     rate = div64((tp->snd_cwnd * tp->mss_cache * 16), tp->rtt_inst);
6
7     if (rate >= max_rate) {
8        max_rate = rate;
9        bestsk = sk;
10    }
11 }
```

A.4 Gerenciador *Single-mesh*

A implementação de *Single-mesh* ocorre em trecho interno de *Full-mesh*, no momento do estabelecimento dos sub-fluxos. A operação básica do *Full-mesh* é procurar toda possível combinação dos endereços de origem com os endereços de destino. Para achar os caminhos é modificado incrementalmente a variável `bitfield`, a cada modificação, o *Full-mesh* tenta estabelecer um sub-fluxo entre o endereço de origem e o endereço de destino gerado através da variação. O Código A.4 apresenta o trecho principal desta implementação. A variável booleana `path_exist` auxilia na identificação do sub-fluxo, se o mesmo já percorre ou não um caminho conhecido. A linha 3, a macro `mptcp_for_each_sk()` executa um laço para percorrer todos os sub-fluxos estabelecidos na conexão MPTCP. Para cada sub-fluxo `sk`, a condição da linha 4 compara se já existe um fluxo estabelecido com mesmo endereço IP de origem de `sk`. Caso exista, assume-se que há um sub-fluxo para aquele caminho e, então interrompe o laço. Caso não exista nenhum sub-fluxo neste caminho, um novo sub-fluxo é criado e estabelecido por `mptcp_init4_subsockets()` e `mptcp_v4_subflows()` inicia a operação do sub-fluxo. Existem funções distintas para quando o MPTCP utilizar o IPv6, como o código é similar, este não foi apresentado no código abaixo.

Código A.4 – Implementação de SM no sub-módulo de gerenciamento *Full-mesh* do MPTCP.

```
1  path_exist = false;
2
3  mptcp_for_each_sk(mpcb, sk) {
4     if(inet_sk(sk)->inet_saddr == mptcp_local->locaddr4[i].addr.s_addr){
5        path_exist = true;
```

```
6         break;
7      }
8  }
9
10 if(path_exist == false){
11    mptcp_init4_subsockets(meta_sk, &mptcp_local->locaddr4[i], &rem4);
12    mptcp_v4_subflows(meta_sk, &mptcp_local->locaddr4[i], &rem4);
13 }
```